Departamento de Informática
Área de Arquitectura y Tecnología de Computadores

PROYECTO FIN DE CARRERA

Diseño de un Simulador de Dinámica Molecular basado en CORBA

Autor: Fco. Javier Aliseda Casado

Tutor: David Expósito Singh

Leganés, Febrero de 2011

Título: Diseño de un Simulador de Dinámica Molecular basado en CORBA
Autor: Fco. Javier Aliseda Casado
Director: David Expósito Singh

EL TRIBUNAL

Presidente: Javier Fernández Muñoz

Vocal: Laura Prada Camacho

Secretario: José Luis López Cuadrado

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 10 de Febrero de 2011 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Quiero mostrar mis agradecimientos, en primer lugar, a mi tutor David, por ofrecerme la posibilidad de desarrollar este interesante proyecto, su constante ayuda y colaboración, y su paciencia en la espera de su conclusión.

En segundo lugar, agradecer a Oscar y Jaime, su colaboración en la realización de los experimentos necesarios para la evaluación del rendimiento sobre el cluster del laboratorio de informática.

También quiere recordar y agradecer a todos mis compañeros y profesores, su apoyo durante todos estos años de carrera, en ocasiones difíciles, pero que sin duda, reconozco me han servido para ampliar en gran medida mis conocimientos, y no solo a nivel teórico sino también a un nivel práctico y cercano a la realidad profesional.

Por último, agradecer a mi mujer Victoria, y a toda mi familia y amigos, por su apoyo en estos años de estudio que finalmente concluyen con este proyecto.

¡ Muchas gracias a todos !

Resumen

La dinámica molecular es una técnica para la simulación por ordenador de la interacción entre átomos y moléculas durante un período de tiempo. Estas simulaciones conllevan un alto consumo de recursos, tanto de memoria como de capacidad de procesamiento. Es por tanto una situación ideal para la utilización de entornos HPC (High-performance computing).

Actualmente, GROMACS es una de las aplicaciones más utilizadas por universidades y centros de investigación para este tipo de simulaciones. Permite realizar simulaciones tanto en secuencial como en paralelo, sobre equipos multiprocesador o redes locales de ordenadores con la misma arquitectura utilizando MPI.
Dada la amplia utilización de esta aplicación hemos considerado interesante poder ejecutar estas simulaciones sobre maquinas con distintas arquitecturas y distribuidas en redes de área extensa, permitiendo así la ejecución en arquitecturas heterogéneas y distintos sistemas operativos.

Para conseguirlo, hemos comenzado este proyecto con el diseño e implementación de una librería de comunicaciones en CORBA, estándar de la OMG para la interconexión de objetos distribuidos con una amplia difusión en la actualidad.
Posteriormente, hemos utilizado esta librería para reemplazar las funciones de comunicaciones de GROMACS y conseguir así generar una nueva versión interoperable.

Completado el desarrollo, se ha procedido a la evaluación del rendimiento de esta nueva versión de GROMACS, con numerosos experimentos con distintos datos de entrada (simulaciones de la molécula Speptide y el virus Hepatitis C) y sobre distintas plataformas (equipo local multinucleo y cluster con 16 nodos).

Fruto de la evaluación de esta nueva versión, la principal conclusión de este proyecto es que, en el marco de las plataformas empleadas y para las simulaciones consideradas, la implementación basada en CORBA no resulta viable dado que introduce una importante sobrecarga en las comunicaciones, la cual limita el rendimiento y escalabilidad de la implementación paralela de GROMACS.

Palabras clave: simulador, dinámica molecular, GROMACS, MPI, CORBA.

Abstract

Molecular dynamics is a technique for computer simulation of interaction between atoms and molecules in a period of time. These simulations involve a high consumption of resources, both memory and CPU. Therefore, these applications are specially suited for HPC (High-performance computing) environments.

Currently, GROMACS is one of the most used applications by universities and research centers for these of simulations. Simulations can be executed in sequential or parallel. Parallel executions use MPI for performing the data distribution. Communications are a critical issue in the overall program performance. One important limitation of MPI is that the compute nodes used by the program have to have the same architecture, i.e., they execute the same binary file.
Given the wide use of this application, we have considered interesting to enable these simulations on machines with different architectures and distributed over wide area networks, allowing them to run on heterogeneus architectures and/or operative systems.

To achieve it, we started this project with the design and implementation of a CORBA−based communications library, the OMG standard for distributed objects interface widely distributed at the present.
Subsequently, we have used this library for replacing the MPI communications functions inside GROMACS by CORBA functions in order to generate a new interoperable version.

Once the development finished, we proceed to evaluate the performance of this new version of GROMACS. We performed many experiments using different input data (simulations of the Speptide molecule and Hepatitis C virus) and we tested if over differents platforms (local machine with a multi-core processor and a computer cluster with 16 nodes).

As result of the evaluation of this new version, the main conclusion of this project is that, within the framework of the platforms used and the simulations considered, the CORBA−based implementation is not feasible because it introduces a significant overhead in communications, which limits the performance and scalability of the parallel implementation of GROMACS.

Keywords: simulator, molecular dynamics, GROMACS, MPI, CORBA.

Índice general

1. INTRODUCCIÓN Y OBJETIVOS 1
1.1. Introducción a la Dinámica Molecular 2
1.2. Objetivos 4
1.3. Fases del desarrollo 5
1.4. Medios empleados 6
1.5. Estructura de la memoria 7
2. GROMACS 9
2.1. Introducción a la Dinámica Molecular 10
2.2. Características de GROMAS 13
2.3. Instalación de GROMACS 15
2.4. Generación de ficheros para la simulación 17
2.5. Ejecución de simulaciones 20
2.6. Profiling y flujo de ejecución 21
2.7. Instalación y ejecución de GROMACS en paralelo (MPI) 24
2.8. Optimización para la ejecución de GROMACS en paralelo 26
3. IMPLEMENTACIÓN GROMACS CON SOPORTE DE INTEROPERABILIDAD. 29
3.1. Requisitos UML 30
3.2. Arquitectura del sistema 32
3.3. Implementación de funciones básicas de comunicación 34
3.3.1 Interfase CORBANetwork 35
3.3.2 Estados de las comunicaciones 36
3.3.3 Rutina de inicialización 37
3.3.4 Rutinas de comunicación/intercambio de datos 39
3.3.5 Rutinas de recopilación de datos 46
3.4. Parámetros de configuración 50
3.5. Optimizaciones realizadas 51
3.5.1 Disminución de los tiempos de espera 51
3.5.2 Mecanismo de reconocimiento de envío/recepción 52
4. ESTUDIO COMPARATIVO DE RENDIMIENTO 55
4.1. Introducción 56
4.1.1 Equipo local con doble núcleo 57
4.1.2 Cluster Kasukabe del laboratorio de informática. 58
4.2. Estudio de la molécula Speptide 62
4.2.1 Generación de la molécula 62
4.2.2 Comparativa de tiempos en equipo local 66
4.2.3 Comparativa de tiempos en cluster Kasukabe 69
4.2.4 Resumen comparativa de tiempos 72
4.2.5 Comparativa de rendimiento en operaciones por segundo 73
4.2.6 Comparativa de consumo de memoria RAM 74
4.3. Estudio del virus de la Hepatitis C 75
4.3.1 Generación del virus 75
4.3.2 Comparativa de tiempos en equipo local 76
4.3.3 Comparativa de tiempos en cluster Kasukabe 79
4.3.4 Resumen comparativa de tiempos 82

4.4. Comparativa MPICH vs ORBit 2 85
5. CONCLUSIONES Y TRABAJOS FUTUROS 87
5.1. Conclusiones 88
5.2. Trabajos futuros 89
6. PRESUPUESTO 91
6.1. Introducción 92
6.2. Estimación de tiempos 93
6.3. Presupuesto 95
6.4. Oferta 97
7. GLOSARIO 99
8. REFERENCIAS 101
9. ANEXO I. MANUAL DE USUARIO 103
Instalación 104
Generación ficheros simulación 106
Ejecución en equipo local 107
Ejecución en cluster Kasukabe 108

Índice de figuras

Figura 1: Ejemplo de simulación del ADN 10
Figura 2: Simulación molécula Speptide 63
Figura 3: Speptide. Tiempos en local con 1 proceso 66
Figura 4: Speptide. Tiempos en local con 2 procesos 67
Figura 5: Speptide. Tiempos en local 2 procesos. Normalizado 67
Figura 6: Speptide. Tiempos en local con 16 procesos 68
Figura 7: Speptide. Tiempos en local con 16 nodos. Normalizado 68
Figura 8: Speptide. Tiempos en cluster usando 1 nodo 69
Figura 9: Speptide. Tiempos en cluster usando 2 nodos 70
Figura 10: Speptide. Tiempos en cluster usando 2 nodos. Normalizado 70
Figura 11: Speptide. Tiempos en cluster usando 16 nodos 71
Figura 12: Speptide. Tiempos en cluster usando 16 nodos. Normalizado 71
Figura 13: Speptide. Comparativa de tiempos 72
Figura 14: Speptide. Comparativa de rendimiento en operaciones por segundo. 73
Figura 15: Speptide. Comparativa consumo memoria RAM 74
Figura 16: Simulación molécula Hepatitis C 76
Figura 17: Hepatitis C. Tiempos en local con 1 proceso 77
Figura 18: Hepatitis C. Tiempos en local con 2 procesos 77
Figura 19: Hepatitis C. Tiempos en local con 2 procesos. Normalizado 78
Figura 20: Hepatitis C. Tiempos en local con 16 procesos 78
Figura 21: Hepatitis C. Tiempos en local con 16 procesos. Normalizado 79
Figura 22: Hepatitis C. Tiempos en cluster con 1 nodo 80
Figura 23: Hepatitis C. Tiempos en cluster con 2 nodos 80
Figura 24: Hepatitis C. Tiempos en cluster con 2 nodos. Normalizado 81
Figura 25: Hepatitis C. Tiempos en cluster con 16 nodos 81
Figura 26: Hepatitis C. Tiempos en cluster con 16 nodos. Normalizado 82
Figura 27: Hepatitis C. Comparativa de tiempos 83
Figura 28: Rendimiento MPICH vs ORBit 86

Índice de diagramas

Diagrama 1: Esquema de preparación ficheros simulación GROMACS 17
Diagrama 2: Flujo de ejecución simulación GROMACS 23
Diagrama 3: Casos de uso 31
Diagrama 4: Arquitectura del sistema 33
Diagrama 5: Estados de las comunicaciones 36
Diagrama 6: Actividad CORBA_Init 38
Diagrama 7: Actividad CORBANetwork_Inicialize 39
Diagrama 8: Actividad CORBA_Send 40
Diagrama 9: Actividad CORBANetwork_SET 41
Diagrama 10: Actividad CORBA_Recv 43
Diagrama 11: Actividad CORBANetwork_GET 44
Diagrama 12: Actividad CORBA_Sendrecv 46
Diagrama 13: Actividad CORBA_Allreduce 48
Diagrama 14: Actividad CORBANetwork_Barrier 49
Diagrama 15: Cluster Kasukabe 60
Diagrama 16: Proyecto. Diagrama de Gantt 94

Índice de tablas

Tabla 1: Estados de las comunicaciones 37
Tabla 2: Parámetros de configuración 50
Tabla 3: Pámetros trabajos gestor de colas 59
Tabla 4: Comandos gestor de colas 60
Tabla 5: Colas del gestor Kasukabe 60
Tabla 6: Cómputo de tiempos en GROMACS 65
Tabla 7: Proyecto. Estimación de tiempos 93

Capítulo 1

Introducción y objetivos

1.1. Introducción a la Dinámica Molecular

La dinámica molecular es una técnica de simulación por ordenador en la que átomos y moléculas interactúan durante un período de tiempo por aproximaciones de la física conocida, dando una visión del movimiento de las partículas. Estas simulaciones se utilizan con frecuencia en el estudio de proteínas y biomoléculas, así como en ciencia de materiales.

Podríamos describir estas simulaciones por ordenador como una técnica de "microscopio virtual" con alta resolución temporal y espacial. Considerando que es posible tomar instantáneas de las estructuras, que no podrían tomarse en experimentos convencionales y permitiendo el acceso a todas las escalas de tiempo y movimiento con resolución atómica.

Por lo tanto, la dinámica molecular permite a los científicos escudriñar en el movimiento de los átomos individuales de un modo que no es posible en experimentos de laboratorio.

Para realizar estas simulaciones, es necesaria la representación en el ordenador de todos los átomos y moléculas que intervienen en el proceso. Estas simulaciones conllevan un gran consumo de recursos, tanto de memoria como de capacidad de procesamiento. Es por tanto habitual, la utilización de ordenadores de gran rendimiento y en muchas ocasiones con grandes capacidades para la computación paralela o distribuida.

Una de las técnicas más usadas en computación paralela es MPI (Message Passing Interface ó Interfaz de Paso de Mensajes), un estándar que se estableció en 1994 (MPI-1) y que define las funciones necesarias para el paso de mensajes entre múltiples procesadores. Esta técnica permite la programación concurrente con sincronización entre procesos y exclusión mutua sin necesidad de un área de memoria compartida.
Esta técnica tiene como principales ventajas su estandarización y portabilidad en equipos multiprocesadores, multicomputadores e incluso redes heterogéneas, con unas buenas prestaciones y con múltiples implementaciones, incluidas versiones libres como Mpich o Lam-mpi.

La principal desventaja de MPI está en su poca flexibilidad, dado que su interfaz está preestablecido por el estándar y sus dificultades para la interoperabilidad entre distintas plataformas.

Otras de las técnicas más utilizadas hoy en día para la computación distribuida es CORBA (Common Object Request Broker Adapter ó Arquitectura común de intermediarios en peticiones de objetos), un estándar que establece una plataforma para el desarrollo de sistemas distribuidos facilitando la invocación de métodos de objetos remotos.
CORBA fue definido por el Object Management Group (OMG) como un protocolo de comunicaciones y los mecanismos necesarios para permitir la interoperabilidad entre diferentes aplicaciones escritas en diferentes lenguajes y ejecutadas en diferentes plataformas.

Estos mecanismos de interoperabilidad conllevan un software adicional (middleware) que se encarga de envolver (wrapper) las llamadas a objetos remotos gestionando las comunicaciones y conversiones de datos entre las distintas máquinas o procesadores.

La principal ventaja, por tanto de CORBA, es la flexibilidad de su interfaz y su interoperabilidad entre distintas plataformas.
En su contra, suele tener un peor rendimiento frente a otras técnicas debido fundamentalmente al middleware que requiere para las comunicaciones.

Entre los sistemas más utilizados actualmente para simulaciones de dinámica molecular, se encuentra la aplicación GROMACS (GROningen MAchine for Chemical Simulatios).

GROMACS es una aplicación de alto rendimiento que fue implementada originalmente en la Universidad de Groningen y que posteriormente ha sido re-implementada como código abierto, por un amplio número de desarrolladores.

Actualmente GROMAS permite realizar simulaciones utilizando MPI con un buen rendimiento en ejecuciones dentro de una misma máquina o en redes de ordenadores con la misma arquitectura.

Dada la amplia utilización de esta aplicación y las ventajas que tendría la posibilidad de ejecutar simulaciones en múltiples máquinas con distintas arquitecturas, nos hemos propuesto con este proyecto, la implementación de una versión de GROMACS interoperable.

1.2. Objetivos

En la actualidad, el diseño de aplicaciones distribuidas se ve facilitado en gran medida por el empleo de sistemas basados en objetos distribuidos. En este contexto, una de las arquitecturas más ampliamente difundidas es CORBA.

El objetivo de este proyecto es evaluar las posibilidades del empleo de CORBA para la computación de alto rendimiento, con el fin de conseguir reducir (mediante computación paralela) el tiempo de ejecución de una simulación de dinámica molecular, y al mismo tiempo conseguir la interoperabilidad entre plataformas heterogéneas.

La aplicación objetivo para esta evaluación es GROMACS, la cual es ampliamente utilizada en entornos de HPC debido al gran número de recurso consumidos.

Como primer paso del proyecto, se ha investigado la posibilidad de que ya existiese alguna implementación de GROMACS con ejecución en paralelo utilizando CORBA, o la existencia de otros simuladores de dinámica molecular con esta posibilidad, y no se ha encontrado ninguna referencia.

Por lo tanto, con este proyecto pretendemos:

1. Reemplazar el código relacionado con las comunicaciones en MPI por una nueva versión en CORBA que nos ofrezca interoperabilidad entre plataformas heterogéneas.

2. Evaluar el rendimiento de GROMACS en esta nueva implementación de las comunicaciones.

3. Analizar las ventajas y/o inconvenientes de la implementación con CORBA versus la versión actual en MPI.

1.3. Fases del desarrollo

El desarrollo del proyecto se ha desarrollado en varias fases que detallamos a continuación:

1) Estudio de GROMAS, generación de simulaciones y sistema de intercomunicación entre nodos en ejecuciones en paralelo.

2) Análisis de implementaciones existentes de CORBA y selección de la más adecuada para nuestro desarrollo en función de sus posibilidades de integración con GROMACS. Finalmente seleccionamos ORBit 2, por ser software libre, código fuente en lenguaje C, y disponibilidad de documentación.

3) Diseño e implementación de una librería de comunicaciones CORBA que mantenga un interfase similar a MPI para permitir su integración en GROMACS.

4) Modificación del código de GROMACS para utilizar la nueva librería.

5) Primeros experimentos sobre distintas plataformas (equipo local multinucleo y cluster de 16 nodos del laboratorio de informática).

6) Estudio comparativo de resultados de los primeros experimentos. Se observa un rendimiento inferior a MPI, por lo que decidimos implementar optimizaciones en el código de la primera librería.

7) Segunda fase de experimentos sobre distintas plataformas (equipo local multinucleo y cluster de 16 nodos del laboratorio de informática) y con distintos datos de entrada (molécula Speptide y virus de la Hepatitis C).

8) Estudio comparativo de resultados de los nuevos experimentos.

9) Análisis de los resultados de los experimentos y conclusiones.

10) Memoria y documentación del proyecto.

1.4. Medios empleados

Para el desarrollo del proyecto se han utilizado tanto medios personales, como los medios que proporciona la universidad: laboratorio de informática y biblioteca principalmente.

Para el desarrollo de la librería, la modificación del código de GROMACS y las pruebas en local, he utilizado mi ordenador personal, con las siguientes características:

- Procesador: Core Duo T5200 (doble núcleo)
- Frecuencia: 1.6 GHz
- RAM: 2.0 GB
- S.O: Linux Ubuntu 9.04 32 bits

Posteriormente, para completar las pruebas, hemos utilizado el cluster Kasukabe del laboratorio de informática compuesto por 16 nodos con las siguientes características:

- Procesador: AMD Athlon XP 1700+
- Frecuencia: 1.1 GHz
- RAM: 1.5 GB
- S.O.: Linux Debian Lenny 32 bits

1.5. Estructura de la memoria

La presente memoria, está estructurada en capítulos y organizados de manera que reflejen las distintas fases del proyecto. A continuación detallamos brevemente el contenido de los mismos:

- Capítulo 2: Este capítulo comienza con una introducción a la dinámica molecular, para posteriormente detallar las características de GROMACS y los pasos necesarios para la realización de simulaciones tanto en secuencial como en paralelo.

- Capítulo 3: Describe el análisis, diseño e implementación de la librería de comunicaciones en CORBA que reemplazará a las llamadas a MPI de GROMACS, sus parámetros de configuración y las optimizaciones realizadas para mejorar su rendimiento.

- Capítulo 4: Realiza un estudio comparativo entre la versión original de GROMACS con MPI y el nuevo desarrollo con CORBA. El estudio se realiza con dos simulaciones, la molécula Speptide y el virus de la Hepatitis C, y sobre distintas plataformas, un equipo local con procesador de doble núcleo y un cluster de 16 nodos.

- Capítulo 5: Analiza los datos obtenidos del estudio anterior para sacar conclusiones y proponer trabajos futuros a desarrollar.

- Capítulo 6: Establece un presupuesto para el presente proyecto, analizando costes y realizando una oferta en consecuencia con los mismos.

Al final del proyecto se incluyen las referencias utilizadas y un glosario.

Como anexo, se incluye el manual de usuario con instrucciones para la instalación de GROMACS, y la generación y ejecución de las simulaciones realizadas en el proyecto.

Capítulo 2

GROMACS

2.1. Introducción a la Dinámica Molecular

GROMACS es un motor para realizar simulaciones de dinámica molecular y minimización de energías. Estas son dos de las muchas técnicas pertenecientes al ámbito de la química computacional y modelización molecular.

La química computacional abarca desde la mecánica cuántica de las moléculas, a la dinámica de los grandes agregados moleculares.

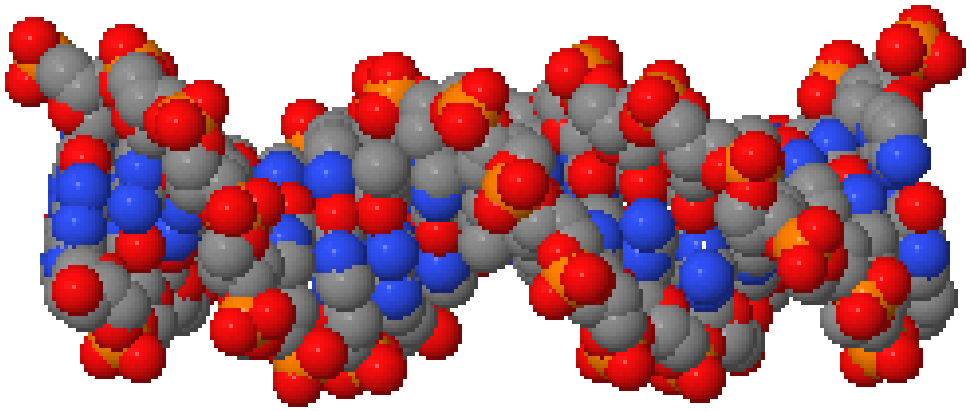


Figura 1: Ejemplo de simulación del ADN

La modelización molecular pretende describir los sistemas químicos complejos en términos de un modelo atómico realista, con el objetivo de entender y predecir las propiedades macroscópicas basada en un conocimiento detallado a escala atómica. A menudo, el modelado molecular se utiliza para diseñar nuevos materiales, para los cuales, una predicción precisa de sus propiedades físicas es muy necesaria.

Las propiedades físicas macroscópicas se pueden distinguir en:

a) Propiedades estáticas en equilibrio, tales como la constante de unión entre un inhibidor y una enzima, la energía potencial media de un sistema, o la función de distribución radial en un líquido.

b) Propiedades dinámicas en desequilibrio, tales como la viscosidad de un líquido, los procesos de difusión en membranas, la dinámica de cambios de fase, las reacciones cinéticas, o la dinámica de defectos en cristales.

La elección de la técnica depende de la cuestión a responder y de la viabilidad del método para obtener resultados fiables en el estado actual de la técnica. Idealmente, con ecuaciones dependientes del tiempo, se describen las propiedades de sistemas moleculares con alta exactitud, pero algo más complejo que el estado en equilibrio de unos pocos átomos no puede ser manejado

en este nivel inicial. Por lo tanto, las aproximaciones son necesarias, cuanto mayor sea la complejidad de un sistema y mientras mayor sea el tiempo del período de los procesos a investigar, más estrictos son los requisitos de las aproximaciones.
En un determinado momento (alcanzado mucho antes de lo que uno quisiera) el enfoque ab-initio debe ser ampliado o reemplazado por un modelo empírico parametrizado.
Cuando las simulaciones basadas en los principios físicos atómicos fallan debido a la complejidad del sistema molecular, el modelado se basa enteramente en análisis de similitud con estructuras conocidas y datos químicos. Los métodos QSAR (Quantitative Structure-Activity Relations) y muchas predicciones basadas en estructuras de proteínas homologadas pertenecen a esta última categoría de simulaciones.

Las propiedades macroscópicas son siempre un promedio sobre un conjunto estadístico representativo (en equilibrio o en desequilibrio) de sistemas moleculares. Para el modelado molecular, esto tiene dos consecuencias importantes:

- El conocimiento sobre una estructura aislada, incluso si es una estructura con una energía global mínima, no es suficiente. Es necesario generar un conjunto representativo a una temperatura determinada, para poder calcular sus propiedades macroscópicas. Pero esto no es suficiente para calcular propiedades en equilibrio termodinámico basadas en energías libres, tales como equilibrios de fase, constantes de enlace, solubilidad, estabilidad relativa de conglomerados moleculares, etc. El cálculo de energías libres y potenciales termodinámicos requiere extensiones especiales de las técnicas de simulación molecular.
- Aunque las simulaciones moleculares proporcionan en principio detalles atómicos de estructuras y sus movimientos, tales detalles no suelen ser relevantes para analizar las propiedades macroscópicas. Esto abre el camino a la simplificación en la descripción de las interacciones y obviar detalles irrelevantes. La ciencia de la mecánica estática proporciona el marco teórico para hacer estas simplificaciones. Existe una jerarquía de métodos que van desde considerar los grupos de átomos como una unidad, describiendo el movimiento en un número reducido de colectivos coordenadas, un promedio superior a las moléculas del disolvente con un potencial de fuerza media combinada con la dinámica estocástica, a la dinámica meseoscópica que describe densidades en lugar de átomos y flujos como respuesta a los gradientes termodinámicos en lugar de velocidades o aceleraciones como respuesta a las fuerzas.

Para la generación de un conjunto representativo en equilibrio, hay dos métodos disponibles:

a) Simulaciones de Monte Carlo
b) Simulaciones de Dinámica Molecular (DM).

Para la generación de conjuntos en desequilibrio y para el análisis de eventos dinámicos, sólo el segundo método es apropiado.
Mientras que las simulaciones de Monte Carlo son más simples que la DM (porque no requieren el cálculo de fuerzas), no ofrecen estadísticas significativamente mejores que la DM en una cantidad determinada de tiempo de computadora. Por lo tanto la DM es la técnica más universal.
Si la configuración de partida está muy lejos del equilibrio, las fuerzas pueden ser excesivamente

grandes, y la simulación DM puede fallar. En estos casos, una fuerte minimización de energía es necesaria. Otra razón para llevar a cabo una minimización de energía, es la eliminación de toda la energía cinética del sistema: si varias instantáneas de simulaciones dinámicas han de compararse, la minimización de energías reduce el ruido térmico y las energías potencias en las estructuras, de modo que se puedan comparar mejor.

2.2. Características de GROMAS

GROMACS es un paquete versátil para realizar simulaciones de dinámica molecular, es decir, simular las ecuaciones de Newton del movimiento para sistemas con cientos de millones de partículas.
Está diseñado principalmente para moléculas bioquímicas como proteínas, lípidos y ácidos nucleicos que contienen complicadas interacciones en condiciones de servidumbre, pero desde GROMACS es extremadamente rápido el cálculo de estas interacciones no enlazantes (que normalmente dominan las simulaciones), y otros muchos grupos que también se están utilizando para la investigación de sistemas no biológicos, por ejemplo, polímeros.

Entre sus características destacaríamos:

- GROMACS es Software Libre, disponible en el marco del GNU General Public License desarrollado por el departamento de Biophysical Chemistry de la Universidad de Groningen.

- Es una aplicación madura. Cuenta con más de 10 años de vida, y se encuentra en constante evolución. Su última versión es del 9 de Noviembre del 2010.

- Es una aplicación compleja, cuenta con más de 700 ficheros de código y cerca de 300.000 líneas de código, con generación automática de código (en C o ensamblador).

- Es ampliamente utilizada por diversas universidades y centros de investigación.

- Soporta todos los algoritmos habituales en dinámica molecular.

- Proporciona un rendimiento extremadamente alto en comparación con programas similares. Incorpora gran cantidad de optimizaciones algorítmicas introducidas en el código; por ejemplo, extrae el cálculo del virial de los lazos más íntimos sobre las interacciones de pares, y utiliza rutinas propias de software para calcular la raíz cuadrada inversa. Se generan automáticamente los bucles en C o Fortran en tiempo de compilación, con optimizaciones adaptadas a la arquitectura. Utiliza instrucciones ensamblador SSE y 3DNow!, instrucciones multimedia para procesadores i386, utiliza todos los registros x86-64 en Opteron x86-64 y máquinas Xeon EM64T. Esto da como resultado un rendimiento excepcional en estaciones de trabajo y PCs de bajo costo.

- Es fácil de usar, con topologías y archivos de parámetros escritos en formato de texto sin cifrar y muestra mensajes claros de error cuando algo va mal. Dado que se utiliza el preprocesador de C, se pueden compilar partes condicionales en función de la arquitectura o incluir otros archivos.

- No hay lenguaje de scripting - todos los programas utilizan una interfaz sencilla con opciones de línea de comandos para archivos de entrada y de salida. Siempre se puede obtener ayuda sobre las opciones mediante el uso de la opción -h.

- A medida que la simulación se lleva a cabo, GROMACS continuamente va informando del punto en que se encuentra la simulación y de la fecha y hora en que espera terminar.

- Los archivos de entrada de trayectorias son independientes de hardware y por lo tanto pueden ser leídos por cualquier versión de GROMACS, incluso si fue compilado utilizando una precisión diferente de coma flotante.

- Se pueden escribir las coordenadas utilizando la compresión con pérdida, lo que proporciona una forma muy compacta de almacenamiento de datos de trayectoria. La exactitud puede ser seleccionada por el usuario.

- Dispone de una amplia selección de herramientas flexibles para el análisis de la salida de la simulación en pantalla (*ngmx*, *xmgrace*, g_energy, etc.)

- El paquete incluye un constructor de topología completamente automatizado para las proteínas, incluso las estructuras de multímeros. Bloques de construcción están disponibles para los 20 residuos de aminoácidos estándar, así como algunas modificada, el nucleótido 4 y 4 deoxinucleotide reside, varios azúcares y lípidos, y algunos grupos especiales como los hemos y varias moléculas pequeñas.

- GROMACS se puede ejecutar en paralelo, utilizando comunicación estándar MPI.

Precisamente esta última característica, la posibilidad de ejecución en paralelo, es la que trataremos de extender con este proyecto.

2.3. Instalación de GROMACS

El paquete GROMACS es distribuido principalmente como código fuente, aunque también se puede instalar como un paquete RPM para Linux.
En la página principal del proyecto (www.gromacs.org) se encuentran toda la información necesaria para la instalación de la aplicación y otros recursos on-line adicionales.
Para la instalación por defecto basta con descargar y descomprimir el paquete y a continuación ejecutar los siguientes comandos:

./configure
make
make install

El script de configuración debería determinar automáticamente las mejores opciones en función de la plataforma en que se esté ejecutando y advertirá si algún componente no se encuentra en el sistema.

GROMACS puede ser compilado en simple o doble precisión. La opción por defecto es simple precisión, pero se puede cambiar fácilmente seleccionando la opción *--enable-float* en el script de configuración. Con doble precisión, las simulaciones podrían ser hasta un 50% más lentas que con simple precisión, dependiendo de la arquitectura del hardware utilizado. Rutinas en ensamblador están incorporadas para simple y doble precisión en procesadores Pentium IV, Opteron e Itanium. En procesadores Power PC con Altivec, Pentium III o Athlon solo es posible la precisión simple debido a limitaciones del hardware. Para el resto de procesadores se utiliza código C o Fortran para los bucles internos de cálculo intensivo.

La portabilidad es el principal objetivo con que GROMACS ha sido diseñado. No obstante, hay unos requisitos mínimos que deben estar presentes:

1) Un sistema operativo tipo UNIX (BSD 4.x o SYSTEM V rev.3 o superior) o sistemas ejecutándose bajo librerías (p.e. CygWin).

2) Un compilador C ANSI.

3) Opcionalmente un compilador Fortran-77 o Fortran-90 para mejorar la rapidez de los bucles internos de cálculo intensivo.

4) Opcionalmente un ensamblador Nasm para utilizar bucles internos en ensamblador para procesadores x86.

Otras características adicionales son chequeadas en el script de configuración y se mostrará una advertencia en el caso de que alguna no se encuentre instalada.

2.4. Generación de ficheros para la simulación

Para realizar una simulación con GROMACS, es necesario generar previamente los ficheros de entrada con los datos de las moléculas que se van a analizar.
Junto con el paquete GROMACS se incluyen un conjunto de programas necesarios para las distintas etapas que conlleva la preparación de estos ficheros de entrada.

En el siguiente diagrama se detallan las etapas típicas de una simulación:

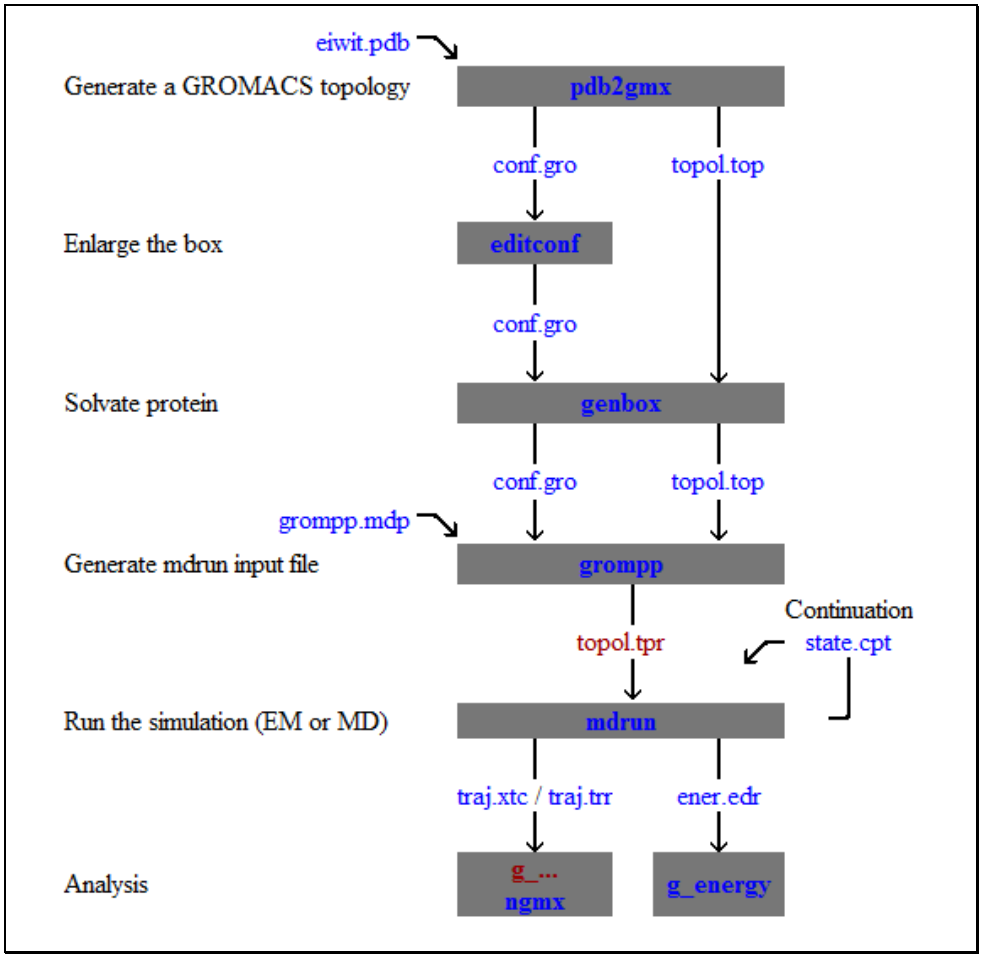


Diagrama 1: Esquema de preparación ficheros simulación GROMACS

Todos los comandos disponen de ayuda utilizando el parámetro *-h*.

GROMACS puede utilizar como entrada ficheros *.pdb* del Protein DataBank (www.pdb.org), para luego convertirlos a su formato nativo *.gro* + *.top* con el comando "*pdb2gmx*". Este

comando toma el archivo *.pdb* y genera dos ficheros, un *.gro* con los átomos y sus coordenadas x-y-z, y un *.top* con las masas atómicas, sus cargas y enlaces.

Una vez que estos ficheros (*.gro* y *.top*) han sido creados, es necesario especificar la "caja" en la que alojar los átomos. Esto se hace ejecutando el comando "*editconf*". Este comando cogerá el fichero *.gro* y lo modificará añadiendo en la última línea las dimensiones de la caja que se le habrán indicado por parámetros.

Ahora que las dimensiones de la caja se han especificado, es posible agregar más de una de las moléculas del fichero *.pdb* y rellenar el resto de la caja con moléculas disolventes. Ambas acciones son realizadas con el comando "*genbox*". Este comando modificará el fichero *.gro* y *.top* para que contengan todos los átomos que han sido agregados en este paso.

En algunas circunstancias, puede ser necesario introducir iones en la solución, ya sea para neutralizar la carga total en el sistema, o para eliminar interacciones electrostáticas entre las moléculas disueltas, o para ambas cosas. Estos iones se agregan al sistema mediante la utilidad "*genion*".

Una vez preparada la solución, hay tres tipos de simulaciones. Las tres son ejecutadas de la misma manera: primero se ejecuta el comando "*grompp*" y a continuación el comando "*mdrun*".

El comando "*mdrun*" es el motor de GROMACS, y el responsable del verdadero desplazamiento de los átomos conforme a las leyes de la física (parametrizadas en el fichero de dinámica molecular *.mdp*).

El comando "*grompp*" toma los ficheros *.gro*, *.top* y *.mdp* y genera un fichero *.tpr* que es la entrada del comando "*mdrun*".

Como salida del comando "*mdrun*" obtenemos un fichero binario *.trr* de gran tamaño, que contiene el estado del sistema a intervalos de tiempo regulares y también un fichero *.gro* que contiene el estado final de la solución. Este fichero *.gro* puede utilizarse para la simulación.

Hay tres tipos de simulaciones, aunque las dos primeras son opcionales, y realmente su función es la preparación de la solución para la simulación tercera, la completa.

La primera simulación tiene como objetivo la minimización de energías (Energy Minimization ó EM) en la solución. Lo único que hace es "empujar" los átomos de las moléculas disueltas hasta que las longitudes de sus enlaces y sus ángulos se encuentren en su configuración de mínima energía potencial (ignorando por completo los átomos de otros elementos de la solución). Hacer esto es una buena manera de comprobar que los enlaces son estables (la molécula podría romperse si no lo fueran), y acelera las simulaciones posteriores.

En algunas soluciones son necesarios varios pasos de minimización de energía (EM), con lo cual será necesario ejecutar varias veces el ciclo *grompp -> mdrun*.

La segunda simulación tiene por objetivo la fijación de la posición del soluto dentro de la disolución. Esto fija las moléculas de soluto en una posición, y permite que el disolvente y los iones se "relajen" a su alrededor en posiciones de mínima energía potencial. Con ello se consigue que el disolvente llene la caja de manera uniforme y no se formen vacíos al inicio de la simulación completa.

La tercera simulación es la completa (full Molecular Dynamics o MD) y en ella se realizan los cálculos completos de la dinámica molecular de la solución en un periodo largo de tiempo (las simulaciones anteriores necesitan mucho menos tiempo en comparación). Esta simulación es la que más tiempo de proceso necesita y es la más indicada para la realización en paralelo.

Como resultado de la simulación se obtienen ficheros *.xtc*, *.trr* y *.edr* que son utilizados para el análisis de la simulación con utilidades como "*ngmx*" o "*xmgrace*", entre otras.

Un esquema completo de todas estas etapas y sus opciones se encuentra en el documento "*GROMACS_MD_Flowchart.pdf*" incluido como referencia.

Para las simulaciones realizadas con este proyecto, los pasos se concretan en el manual de usuario incluido como apéndice del proyecto "Generación ficheros simulación".

2.5. Ejecución de simulaciones

Una vez instalado GROMACS y generados los ficheros necesarios para la simulación, la ejecución en secuencial de simulaciones se realiza con el comando *"mdrun*".
Este comando recibe múltiples parámetros, cuya descripción está documentada en la web de proyecto GROMACS.

Para las simulaciones realizadas con este proyecto, una vez realizados los pasos indicados anteriormente, la llamada al comando sería de la siguiente forma:

mdrun -v -s pr -e pr -o pr -c after_pr -g prlog

Esta llamada se realizaría sobre el directorio donde se ubiquen los ficheros de la simulación.

2.6. Profiling y flujo de ejecución

Para analizar la ejecución de una simulación de GROMACS y obtener datos de rendimiento de las distintas rutinas que se ejecutan procederemos de la siguiente forma:

1) Crear la variable de entorno:

   export CC="gcc -pg"

2) Instalar GROMACS:

   ./configure
   make
   make install

3) Una vez creados los ficheros de la simulación, ejecutar la simulación:

   mdrun -v -s pr -e pr -o pr -c after_pr -g prlog

   Durante la ejecución se generará el fichero de monitorización "*gmon.out*"

4) Visualizar la monitorización con el comando "*gprof*":

   gprof <PATH>/mdrun

Esta utilidad nos va a proporcionar datos como los que se muestran a continuación (extracto de los primeros registros obtenidos de la simulación secuencial de la molécula Speptide en el equipo local):

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 40.24     17.78    17.78                             nb_kernel112_ia32_sse
 13.69     23.83     6.05      501     0.01     0.02  ns5_core
  4.91     26.00     2.17                             nb_kernel100_ia32_sse
  4.47     27.98     1.98     5003     0.00     0.00  csettle
  3.32     29.44     1.47  1806746     0.00     0.00  put_in_list
  3.10     30.81     1.37     5001     0.00     0.00  do_update_md
  2.78     32.04     1.23     5002     0.00     0.00  calc_ke_part
  2.65     33.21     1.17                             nb_kernel010_ia32_sse
  2.44     34.29     1.08                             nb_kernel110_ia32_sse
  2.21     35.27     0.97 29767850     0.00     0.00  invsqrt
  2.06     36.17     0.91                             nb_kernel111_ia32_sse
  1.65     36.91     0.73     5001     0.00     0.00  lo_fcv
  1.20     37.44     0.53 48711373     0.00     0.00  add_j_to_nblist
  1.09     37.91     0.48     5001     0.00     0.00  calc_vcm_grp
  1.04     38.38     0.46                             nb_kernel101_ia32_sse
  1.00     38.81     0.44  7733508     0.00     0.00  get_dx_dd
  0.91     39.22     0.40     5003     0.00     0.00  do_lincs
  0.88     39.60     0.39  1580316     0.00     0.00  nb_kernel330
  0.72     39.92     0.32     5001     0.00     0.00  unshift_x
  0.70     40.23     0.31    10006     0.00     0.00  lincs_matrix_expand
  0.61     40.51     0.27     5002     0.00     0.00  shift_self
  0.61     40.77     0.27     5001     0.00     0.00  calc_mu
  0.60     41.04     0.27  9267813     0.00     0.00  new_i_nblist
  0.48     41.25     0.21     5001     0.00     0.00  do_nonbonded14
  0.45     41.45     0.20  6036207     0.00     0.00  invsqrt
  0.43     41.64     0.19  9267813     0.00     0.00  close_i_nblist
  0.41     41.82     0.18     5001     0.00     0.00  g96angles
  0.38     41.99     0.17   940188     0.00     0.00  do_dih_fup
  0.36     42.15     0.16   932862     0.00     0.00  setexcl
  0.28     42.27     0.12 13707741     0.00     0.00  rvec_dec
  0.27     42.40     0.12     5001     0.00     0.00  do_stopcm_grp
  0.26     42.51     0.12  7471494     0.00     0.00  rvec_sub
  0.25     42.62     0.11  7916583     0.00     0.00  rvec_inc
  0.20     42.71     0.09  2325465     0.00     0.00  cos_angle
  0.18     42.79     0.08  1378723     0.00     0.00  mk_grey
  0.18     42.87     0.08    10002     0.00     0.00  clear_rvecs
```

De esta forma podemos obtener información detallada sobre las rutinas que ejecuta GROMACS durante la simulación, y que se detallan en el siguiente diagrama que describe el flujo de ejecución de una simulación:

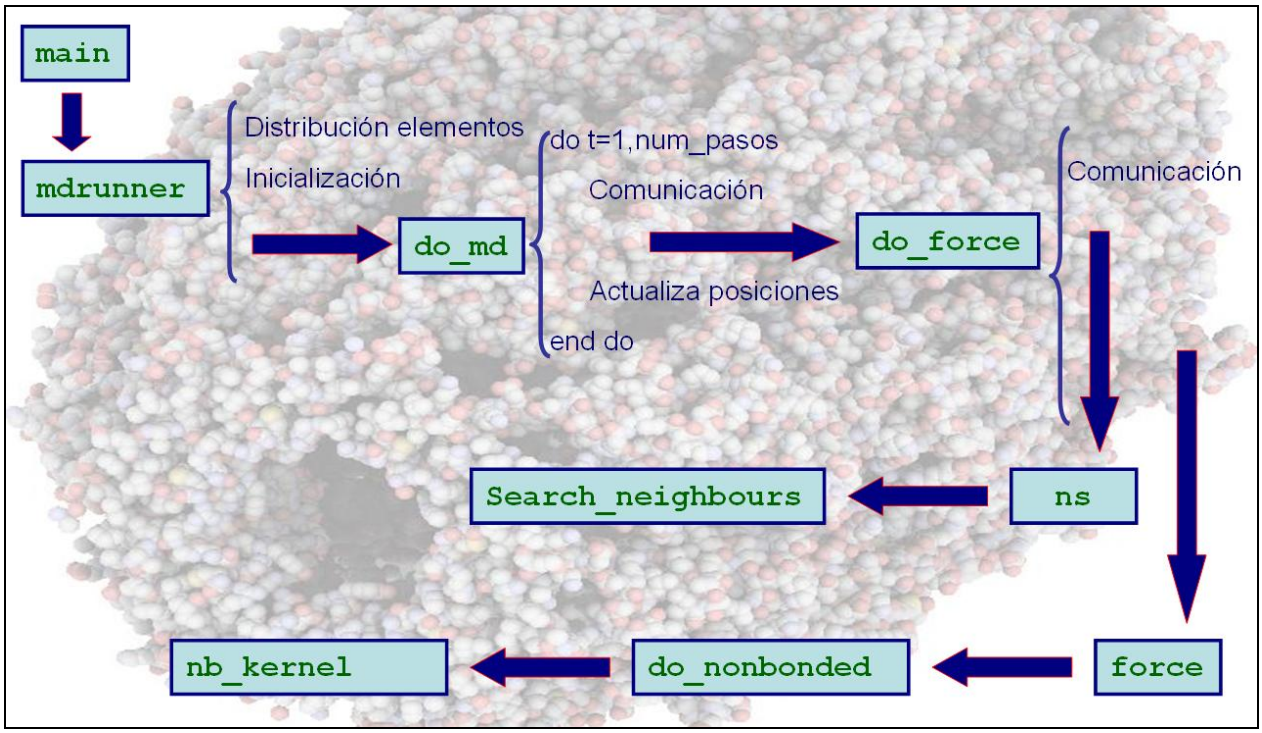


Diagrama 2: Flujo de ejecución simulación GROMACS

Este diagrama, así como el procedimiento para hacer un profiling de GROMACS está extraído del trabajo "Benchmarks!" de mi tutor, David Expósito Singh, cuya referencia se incluye al final del proyecto.

2.7. Instalación y ejecución de GROMACS en paralelo (MPI)

GROMACS permite además realizar simulaciones en supercomputadores, clusters de ordenadores o equipos con multinucleo, haciendo uso de la librería MPI para comunicaciones entre procesos.
Habitualmente, estas librerías se encuentran ya instaladas en supercomputadores, pero también pueden instalarse librerías gratuitas disponibles en Internet para la gran mayoría de plataformas. En el sitio web del proyecto se encuentran enlaces actualizados a estas librerías.

Si tenemos instaladas librerías de MPI en el ordenador u ordenadores en que vayamos a realizar simulaciones, tendremos que compilar GROMACS con la siguiente opción:

./configure --enable-mpi
make
make install

Esta opción permite que se haga uso de MPI para las comunicaciones entre procesos y poder ejecutar la simulación en paralelo.
Si previamente se hubiera configurado GROMACS sin esta opción, es necesario ejecutar previamente:

make distclean

Una vez instalado GROMACS con soporte MPI, la ejecución de simulaciones debe lanzarse con el siguiente comando:

mpirun -np <número de procesos> mdrun <parámetros de la simulación>

2.8. Optimización para la ejecución de GROMACS en paralelo

GROMACS está diseñado de forma que todas las rutinas relacionadas con la comunicación entre procesos paralelos están implementadas en un único fichero fuente. Esto permite implementar una nueva librería específica para el hardware en que se vaya ejecutar la simulación o, como es nuestro caso, implementar librerías basadas en otra interfaz distinta de MPI.

El interfaz de las rutinas de comunicación con el resto del programa está descrito en el fichero:

gromacs/src/gmxlib/network.h

Este interfaz se encuentra implementado en el fichero "*network.c*" y cuenta con las siguientes funciones:

- *extern void gmx_tx(int pid,void *buf,int bufsize);*
  Esta rutina envía, de forma asíncrona, el contenido del buffer (*buf*) de longitud (*bufsize*) al proceso indicado (*pid*).

- *extern void gmx tx_wait(int pid);*
  Esta rutina espera a que el último envío concluya.

- *extern void gmx_txs(int pid,void *buf,int bufsize);*
  Esta rutina envía, de forma síncrona, el contenido del buffer (*buf*) de longitud (*bufsize*) al proceso indicado (*pid*). Al ser síncrona podría "colgar" al emisor si el receptor no confirma su recepción.

- *extern void gmx_rx(int pid,void *buf,int bufsize);*

- *extern void gmx_rx wait(int pid);*

- *extern void gmx_rxs(int pid,void *buf,int bufsize);*
  Estas rutinas realizan la recepción de los datos por el proceso destinatario.

- *extern void gmx_init(int pid,int nprocs);*
  Esta rutina inicializa los dispositivos necesarios para la comunicación entre procesos.

- *extern void gmx_stat(FILE *fp,char *msg);*
  Con esta rutina se puede diagnosticar el estado de las comunicaciones.

En este proyecto, vamos a re-implementar estas rutinas de forma que, en lugar de utilizar el interfaz MPI para la comunicación entre procesos, utilicen una nueva librería implementada específicamente para la utilización de CORBA.

Capítulo 3

Implementación GROMACS con soporte de interoperabilidad.

3.1. Requisitos UML

Para la implementación de GROMACS con soporte de interoperabilidad basados en CORBA ha sido necesaria la re-implementación de las llamadas que actualmente hace a MPI por una versión equivalente en CORBA.
Como ya se explicó con anterioridad, las rutinas de comunicaciones de GROMAS están definidas en el fichero "*network.h*" y codificadas en "*network.c*".

Por lo tanto, los requisitos se han obtenido analizando dichos ficheros y se detallan en el siguiente diagrama de casos de uso:

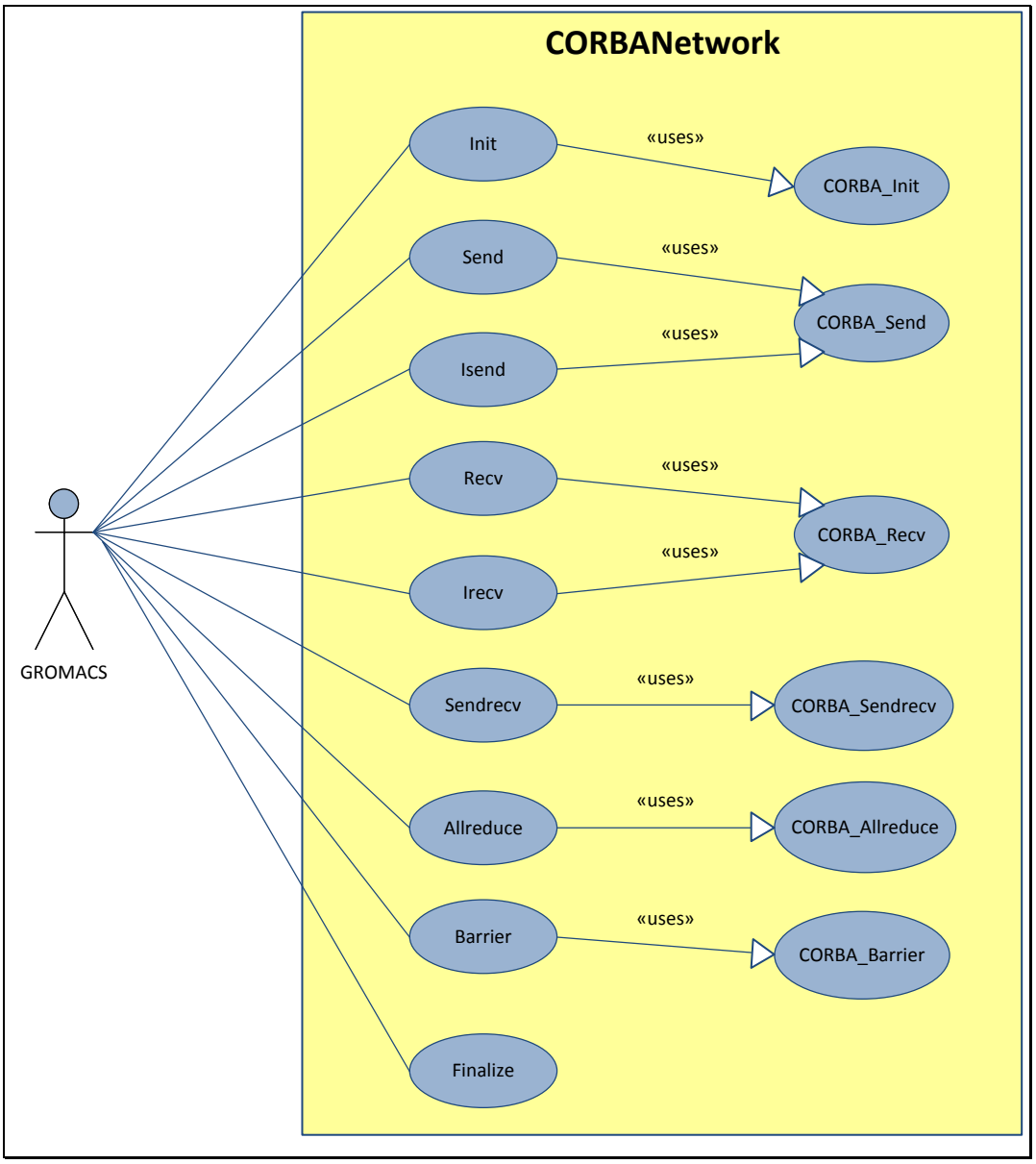


Diagrama 3: Casos de uso

Como se puede observar, el nuevo sistema implementa todas las rutinas de comunicaciones MPI que se utilizan en el GROMACS y que necesitan una re-implementación en CORBA. Para los casos más complejos, se han creado funciones específicas que luego son usadas por varios casos de uso.

3.2. Arquitectura del sistema

Del análisis de los requisitos se ha determinado la conveniencia de desarrollar la librería CORBANetwork ("*libCORBANetworkPFC.so*") que implemente las funciones básicas de comunicación:

- *CORBA_Init*
- *CORBA_Send*
- *CORBA_Recv*
- *CORBA_Sendrecv*
- *CORBA_Allreduce*
- *CORBA_Barrier*

Estas funciones nos servirán para este proyecto, pero también podrían reutilizarse en proyectos similares que pretendan reemplazar MPI por CORBA.

Por otra parte, es necesario modificar el código de GROMACS, concretamente se ha modificado el módulo "*network.c*" de manera que haga uso de las funciones definidas en "*corbanetworkpfc.h*", donde se declaran las llamadas CORBA equivalentes a MPI pero utilizando la librería "*libCORBANetworkPFC.so*".

Tendremos por tanto la siguiente arquitectura:

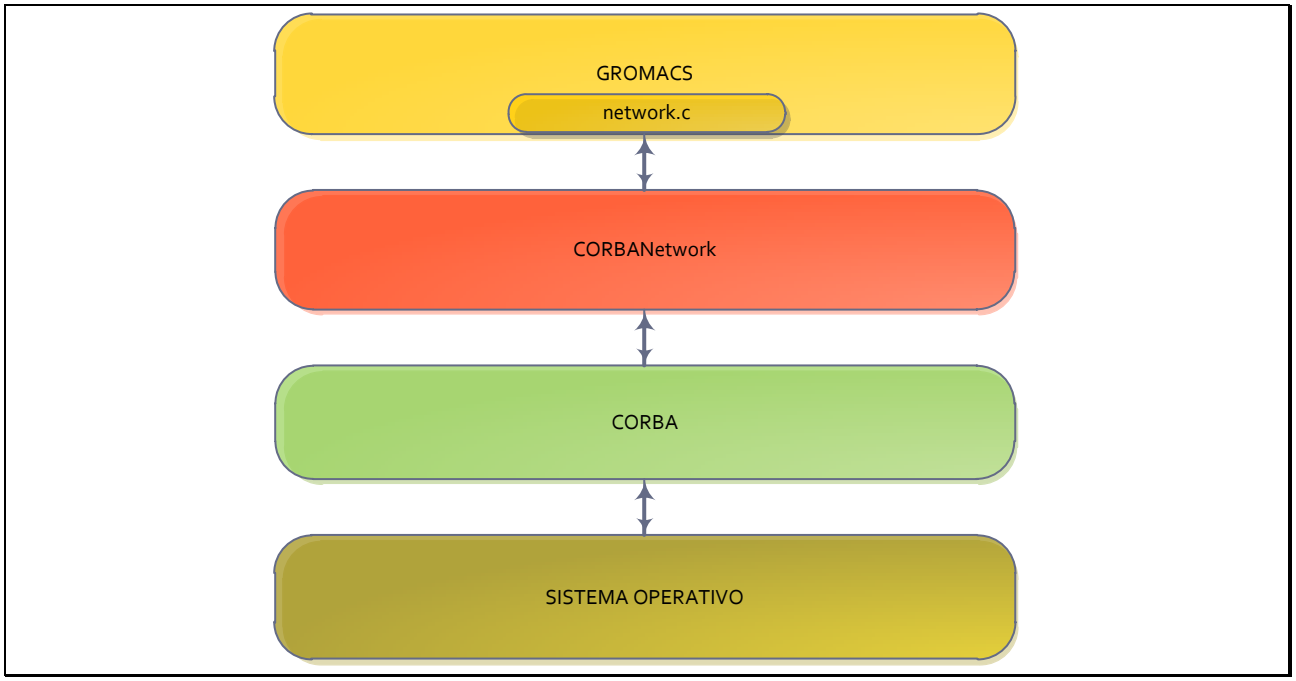


Diagrama 4: Arquitectura del sistema

3.3. Implementación de funciones básicas de comunicación

Las comunicaciones entre los distintos procesos lanzados por la aplicación se dividen en tres grupos:

- Rutina de inicialización.
- Rutinas de comunicación/intercambio de datos.
- Rutinas de recopilación de datos.

Estas rutinas se implementa en el fichero "corbanetworkpfc.c" y definidas en "corbanetworkpfc.h" y realizan las misma función que sus homónimas en MPI.
No obstante, durante la ejecución de estas funciones, en lugar de hacer llamadas a procedimientos de envío/recepción por MPI, se instancia un objeto CORBANetwork de la librería que se ejecutará en segundo plano, y que llevará a cabo la misma función.

Para ello, en cada nodo el objeto CORBANetwork mantendrá una máquina de estados que delimita las distintas situaciones en que puede estar una comunicación entre nodos.

3.3.1 Interfase CORBANetwork

La librería CORBANETWORK está fundamentada en el objeto del mismo nombre, cuya interfase se define a continuación:

```
module corbanetwork {

	// Tipos de buffer:
	typedef sequence<octet>	bufferByte;
	typedef sequence<long>	bufferInt;
	typedef sequence<float>	bufferFloat;
	typedef sequence<double>	bufferDouble;

	interface Storage {

		// Inicializar:
		long initialize (in long numNodes, in long nodeId);

		// Byte:
		long  setByte (in long idSender, in bufferByte buffer);
		bufferByte getByte (in long idSender);

		// Int:
		long  setInt (in long idSender, in bufferInt buffer);
		bufferInt getInt (in long idSender);

		// Float:
		long  setFloat (in long idSender, in bufferFloat buffer);
		bufferFloat getFloat (in long idSender);

		// Double:
		long  setDouble (in long idSender, in bufferDouble buffer);
		bufferDouble getDouble (in long idSender);

		// Barrier:
		long barrier (in long state);

		// Allreduce:
		long allreduce (in long datatype, in long operation);

		};
	};
```

Como se observa, se definen en primer lugar los buffers en función del tipo de datos a manejar. Estos buffers son punteros y no reservan memoria hasta que no se vaya a realizar un envío/recepción.
Por otra parte, dispone de un método de inicialización (inicialize), métodos de envío (set) y recepción (get) en función del tipo de datos, un método para implementar una barrera (barrier) y otro para reducción (allreduce). Estos métodos serán explicados en posteriores apartados.

El resto de funciones MPI no se utilizan en GROMACS, o son reemplazables por una conjunción de las anteriores, como MPI_sendrecv.
En futuros proyectos se podría ampliar esta interfase con el resto de funciones MPI.

3.3.2 Estados de las comunicaciones

A continuación vamos a describir los distintos estados por los que puede pasar una comunicación entre nodos:

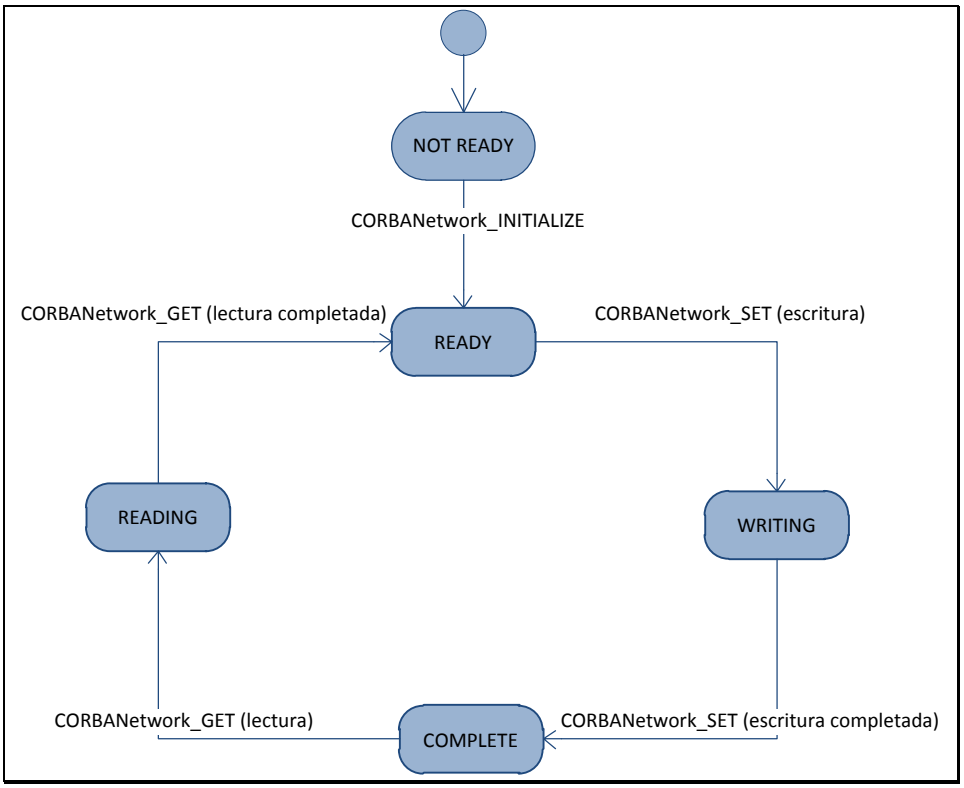


Diagrama 5: Estados de las comunicaciones

Cada nodo dispondrá de un buffer para cada uno de los nodos intervinientes en la comunicación, aunque no se reservará memoria para su uso hasta el momento en que sea necesaria. De igual manera, contará con un flag indicativo del estado del buffer correspondiente a cada nodo.

A continuación se describen cada uno de los estados:

ESTADO	DESCRIPCIÓN
NOT READY	El nodo no está inicializado y por tanto no está preparado ni para enviar (SET) ni para recibir (GET) en ninguno de sus buffers.
READY	El nodo está inicializado y el buffer del nodo correspondiente está disponible (está vacío o se acaba de completar un GET).
WRITING	El buffer del nodo correspondiente está siendo escrito por en emisor (se está ejecutando un SET).
READING	El buffer del nodo correspondiente está siendo leído por el receptor (se está ejecutando un GET).
COMPLETE	La escritura del buffer correspondiente se ha completado (SET completado).

Tabla 1: Estados de las comunicaciones

3.3.3 Rutina de inicialización

La inicialización tiene por objetivo la creación en cada nodo de las estructuras necesarias para llevar a cabo las posteriores comunicaciones utilizando la infraestructura CORBA.
La función que realiza esta labor es CORBA_Init:

```
void CORBA_Init (int num_nodes, int my_rank)
```

Como parámetros recibe:

- *num_nodes:* *Número de nodos lanzados en la ejecución.*
- *my_rank:* *Identificador del nodo que ejecuta la función.*

Su actividad se describe en el siguiente esquema:

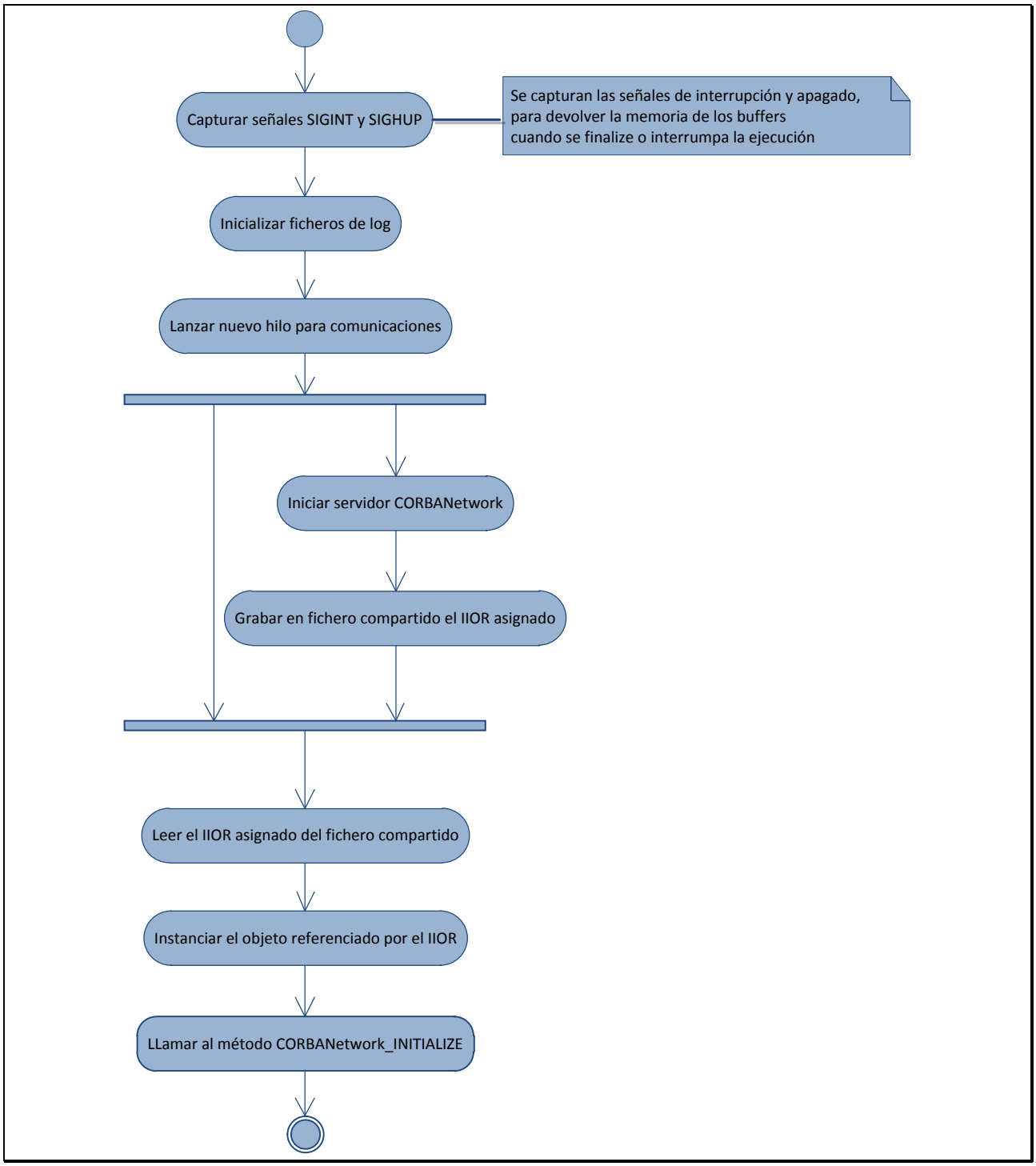


Diagrama 6: Actividad CORBA_Init

El método CORBANetwork_INITIALIZE de la librería CORBANetwork realiza la inicialización de propiedades y buffers siguiendo el esquema:

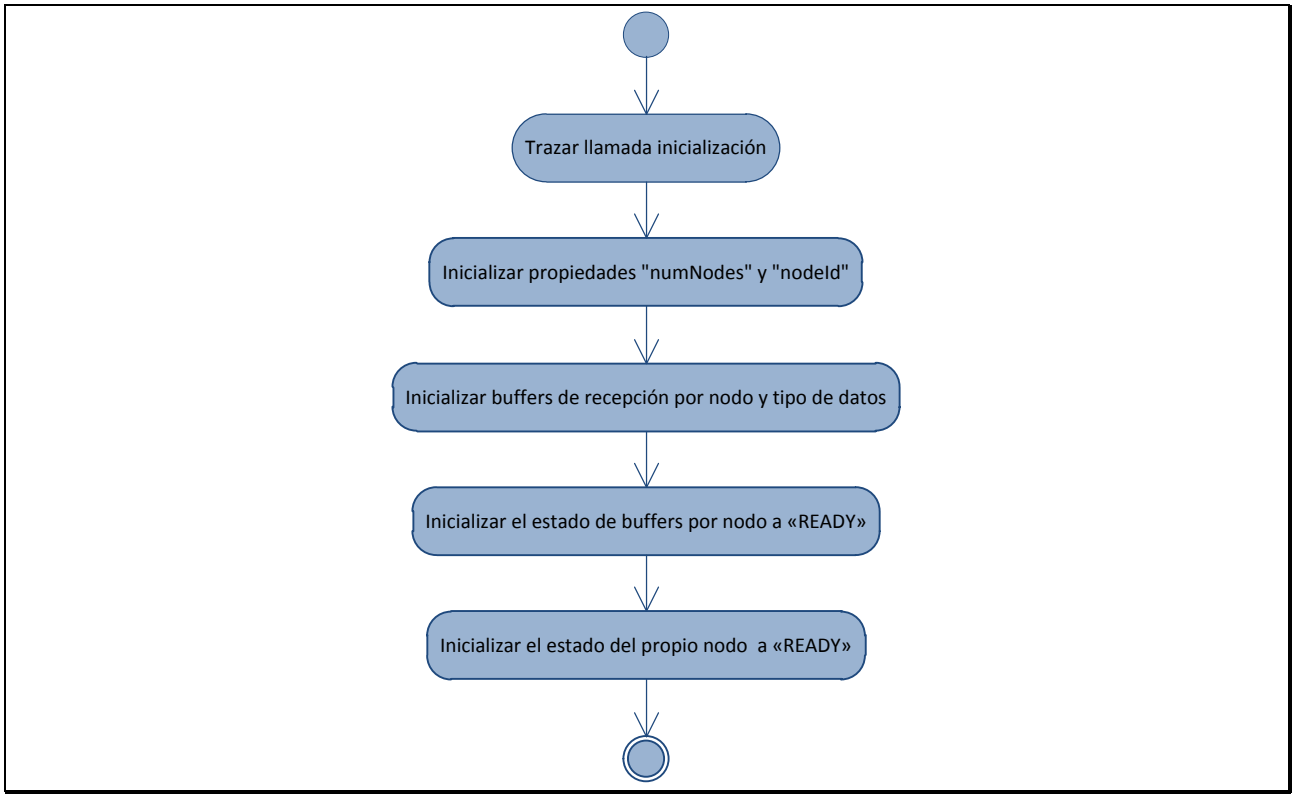


Diagrama 7: Actividad CORBANetwork_Inicialize

3.3.4 Rutinas de comunicación/intercambio de datos

Las rutinas encargadas de las comunicación/intercambio de datos han de realizar las misma funciones disponibles en MPI, pero utilizando CORBA.
MPI dispone de múltiples funciones en función en función del tipo de datos a enviar/recibir y el sincronismo.
Todas las funciones de envío se han englobado en la función CORBA_Send que recibe por parámetros los valores necesarios para realizar el envío.

```
int CORBA_Send (int nodeId, void *buf, int count, int datatype, int dest)
```

Como parámetros recibe:

- *nodeId:* *Identificador del nodo que ejecuta la función.*
- *buf:* *Apuntador al buffer en memoria con los datos a enviar.*
- *count:* *Número de datos a enviar.*
- *datatype:* *Tipo de datos a enviar.*
- *dest:* *Identificador del nodo destinatario de los datos.*

Para minimizar los cambios en el código de GROMACS, el parámetro *datatype* admite los valores definidos por *MPI_Datatype* dentro del fichero "*mpi.h*". Si no se dispone de este fichero, también están definidos los tipos dentro de la propia librería.
Dado que GROMACS no utiliza todos los tipos de datos disponibles en MPI, la librería desarrollada para este proyecto solo implementa el envío de los siguientes tipos de datos:

- *MPI_BYTE*
- *MPI_INT*
- *MPI_FLOAT*
- *MPI_DOUBLE*

El esquema de esta función se describe con el siguiente diagrama de actividad:

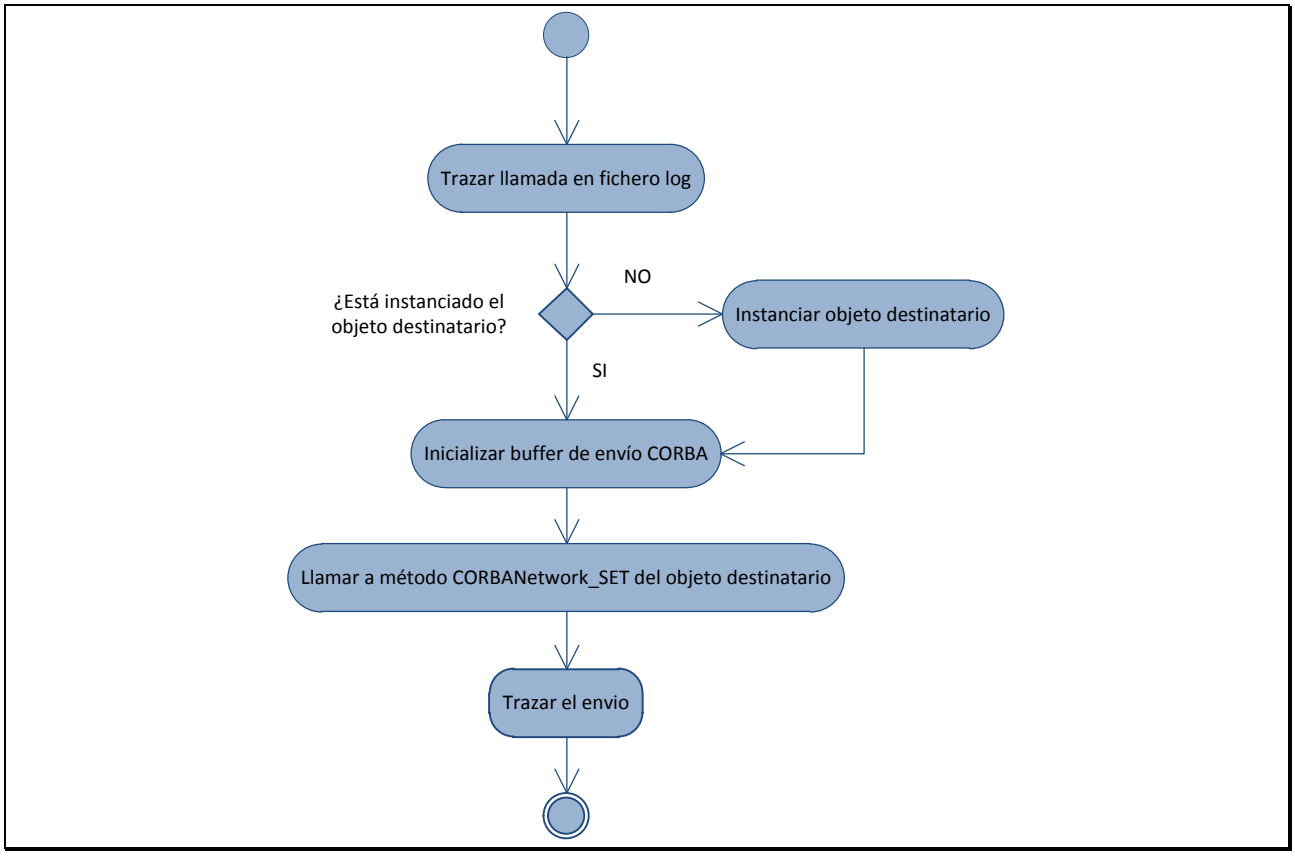


Diagrama 8: Actividad CORBA_Send

El método CORBANetwork_SET de la librería CORBANetwork realiza el envío siguiendo el siguiente esquema:

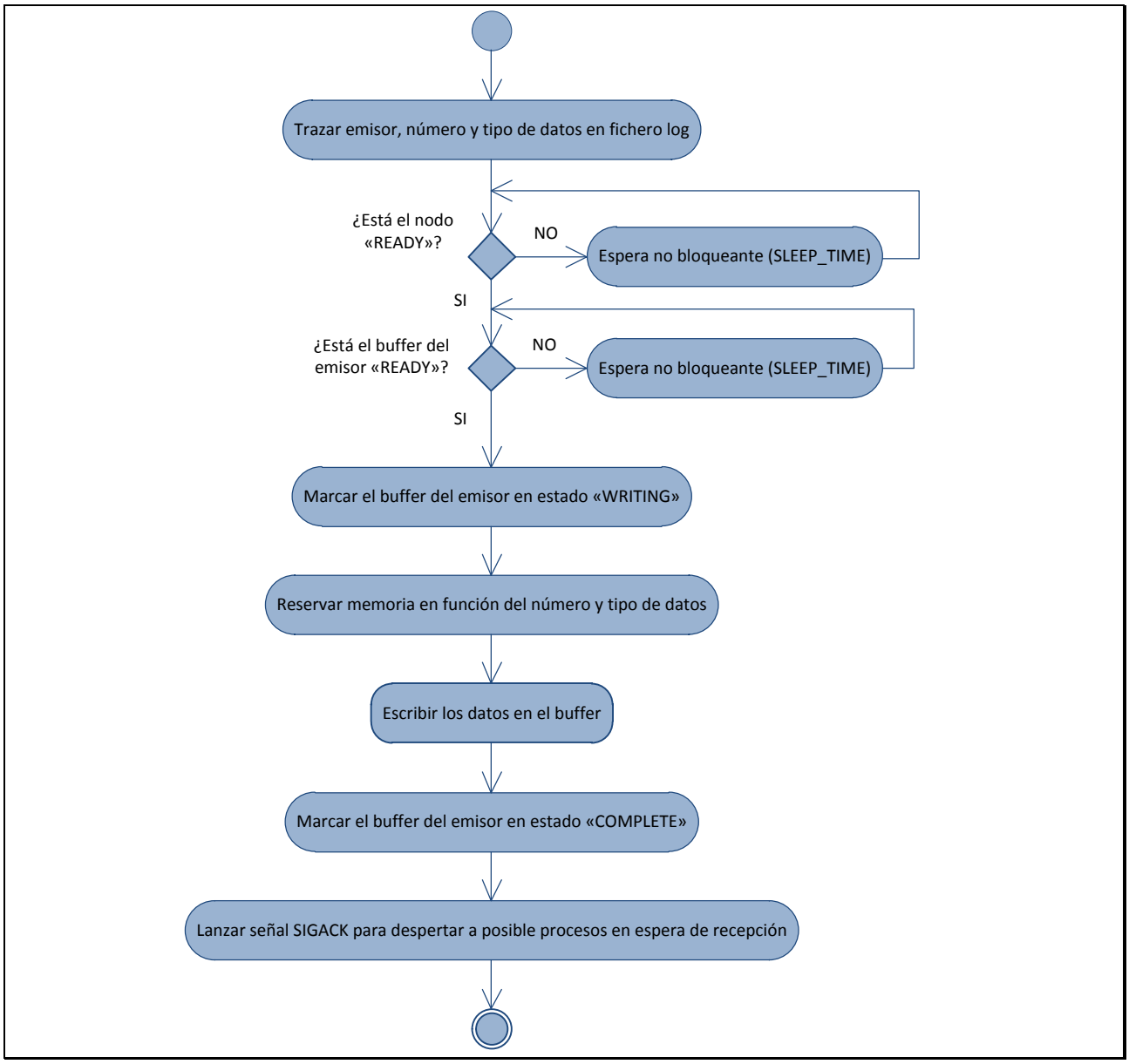


Diagrama 9: Actividad CORBANetwork_SET

Todas las funciones de recepción se han englobado en la función CORBA_Recv que recibe por parámetros los valores necesarios para realizar el envío.

```
int CORBA_Recv (int nodeId, void *buf, int count, int datatype, int dest)
```

Como parámetros recibe:

- *nodeId:* *Identificador del nodo que ejecuta la función.*
- *buf:* *Apuntador al buffer en memoria donde almacenar los datos a recibir.*
- *count:* *Número de datos a recibir.*
- *datatype:* *Tipo de datos a recibir.*
- *dest:* *Identificador del nodo emisor de los datos.*

Como el caso anterior, el parámetro *datatype* admite los valores definidos por MPI_Datatype dentro del fichero "*mpi.h*", y los tipos de datos implementados son los siguientes:

- *MPI_BYTE*
- *MPI_INT*
- *MPI_FLOAT*
- *MPI_DOUBLE*

El esquema de esta función se describe con el siguiente diagrama de actividad:

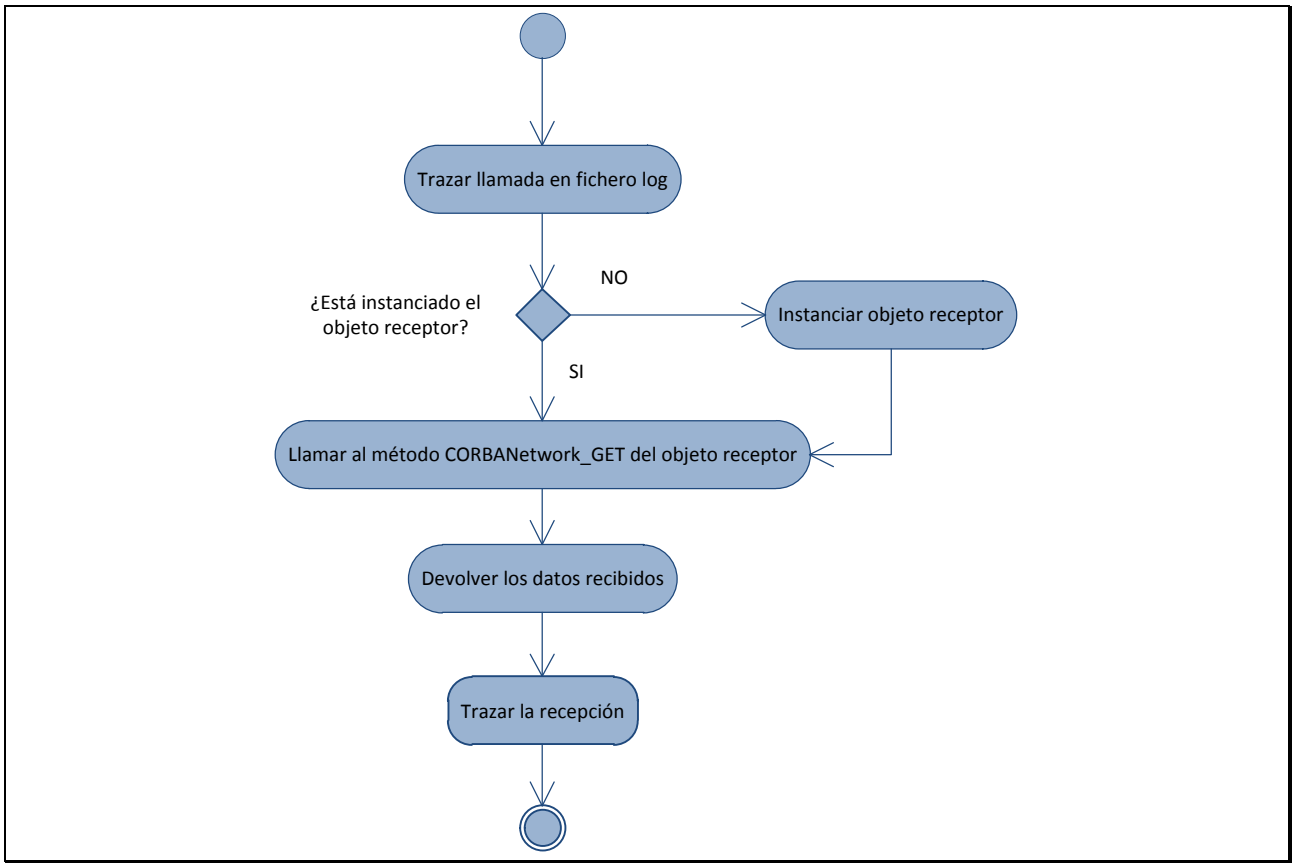


Diagrama 10: Actividad CORBA_Recv

El método CORBANetwork_GET de la librería CORBANetwork realiza el envío siguiendo el siguiente esquema:

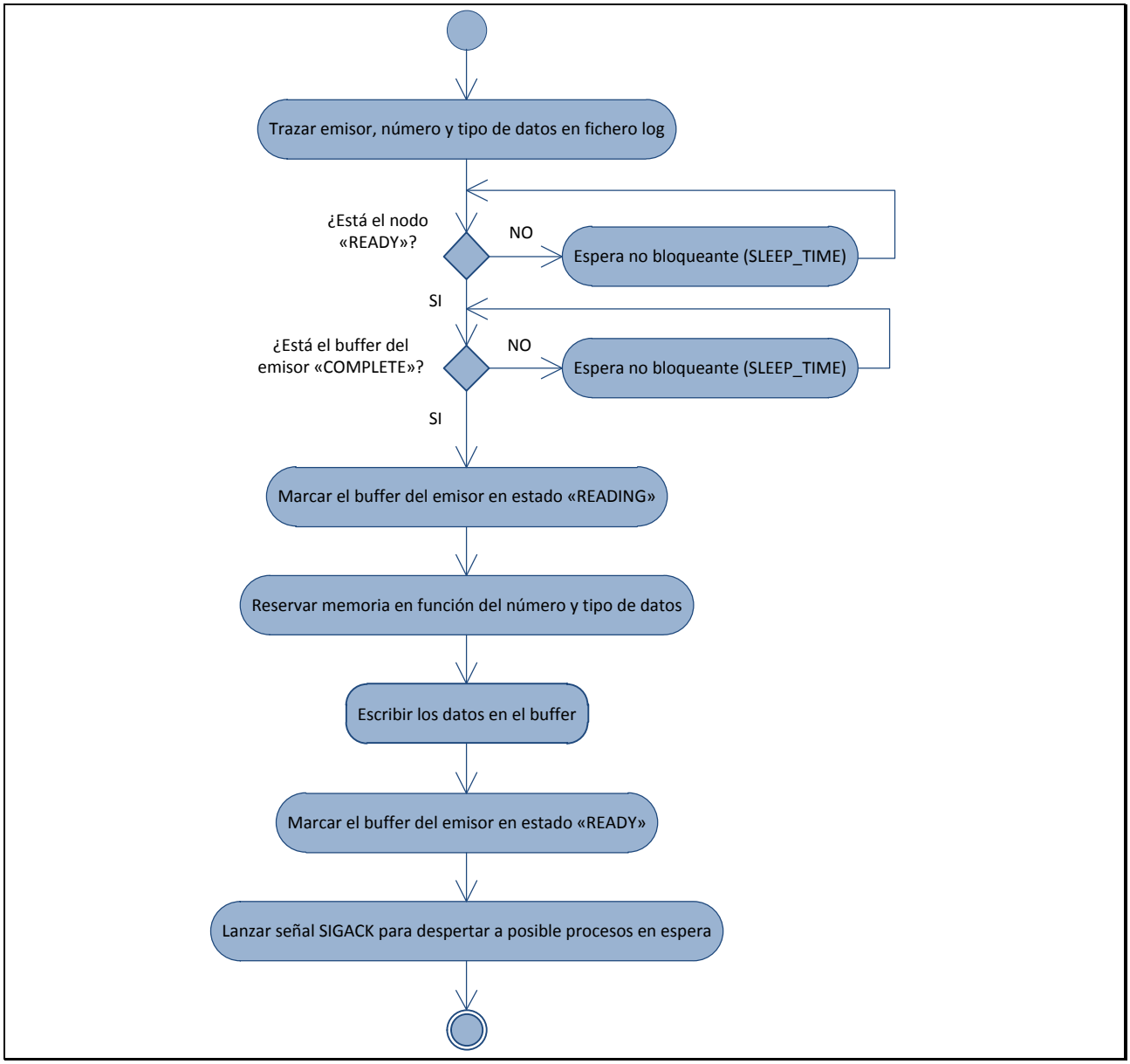


Diagrama 11: Actividad CORBANetwork_GET

Por último la función CORBA_Sendrecv realiza una secuencia de envío y posterior recepción entre dos nodos:

```
int CORBA_Sendrecv (    int nodeId, void *bufsend, int countsend,
                        int datatypesend, int destsend, void *bufrecv,
                        int countrecv, int datatyperecv, int destrecv)
```

Como parámetros recibe:

- *nodeId:* *Identificador del nodo que ejecuta la función.*
- *bufsend:* *Apuntador al buffer en memoria con los datos a enviar.*
- *countsend:* *Número de datos a enviar.*
- *datatypesend: Tipo de datos a enviar.*
- *destsend:* *Identificador del nodo destinatario de los datos.*
- *bufrecv:* *Apuntador al buffer en memoria donde recibir los datos.*
- *countrecv:* *Número de datos a recibir*
- *datatyperecv: Tipo de datos a recibir.*
- *destrecv:* *Identificador del nodo receptor de los datos.*

Como en casos anteriores, el parámetro *datatype* admite los valores definidos por MPI_Datatype dentro del fichero "*mpi.h*", y los tipos de datos implementados son los siguientes:

- *MPI_BYTE*
- *MPI_INT*
- *MPI_FLOAT*
- *MPI_DOUBLE*

El esquema de esta función se describe con el siguiente diagrama de actividad:

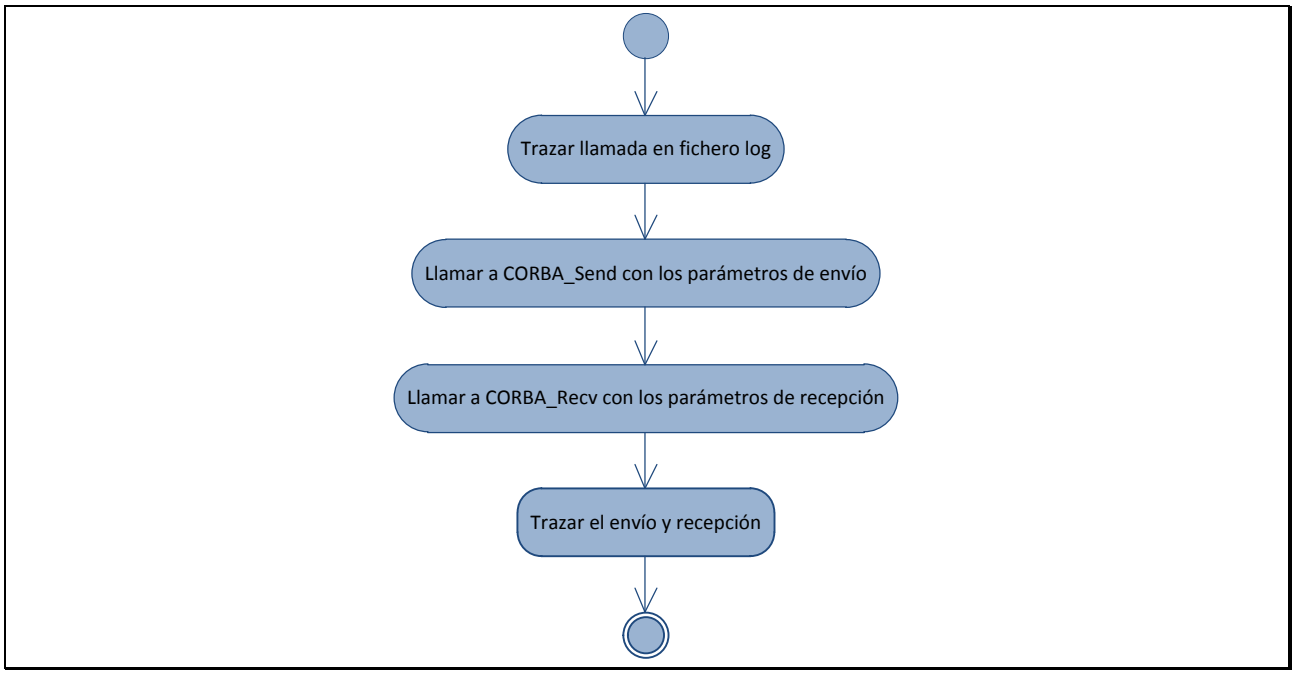


Diagrama 12: Actividad CORBA_Sendrecv

Con estas tres funciones, implementadas para los tipos de datos *byte*, *int*, *float* y *double*, se han cubierto todas las necesidades de intercambio de datos en GROMACS.

3.3.5 Rutinas de recopilación de datos

Dentro de las rutinas de recopilación de datos, hemos tenido que implementar una versión en CORBA del la llamada MPI_Allreduce:

```
int CORBA_Allreduce (int nodeId, int numNodes, void* inbuf, void* outbuf, int count, int datatype, int op)
```

Como parámetros recibe:

- *nodeId:* *Identificador del nodo que ejecuta la función.*
- *numNodes:* *Número de nodos que intervienen en la comunicación.*
- *inbuf:* *Apuntador al buffer en memoria donde albergar los datos recibidos.*
- *outbuf:* *Apuntador al buffer en memoria donde albergar los datos calculados.*
- *count:* *Número de datos a manejar.*
- *datatype:* *Tipo de datos a manejar.*
- *op:* *Operación a realizar.*

Como en las rutinas anteriores, para minimizar los cambios en el código de GROMACS, el parámetro *datatype* admite los valores definidos por *MPI_Datatype* dentro del fichero "*mpi.h*", y el parámetro *op* admite los valores definidos por *MPI_Op* en el mismo fichero. Si no se dispone de este fichero, también están definidos los tipos dentro de la propia librería.

Dado que GROMACS no utiliza todos los tipos de datos disponibles, la librería desarrollada para este proyecto solo implementa el envío de los siguientes tipos de datos:

- *MPI_INT*
- *MPI_FLOAT*
- *MPI_DOUBLE*

Por otra parte, y por el mismo motivo, la única operación implementada ha sido *MPI_SUM*.

El esquema de esta función se describe con el siguiente diagrama de actividad:

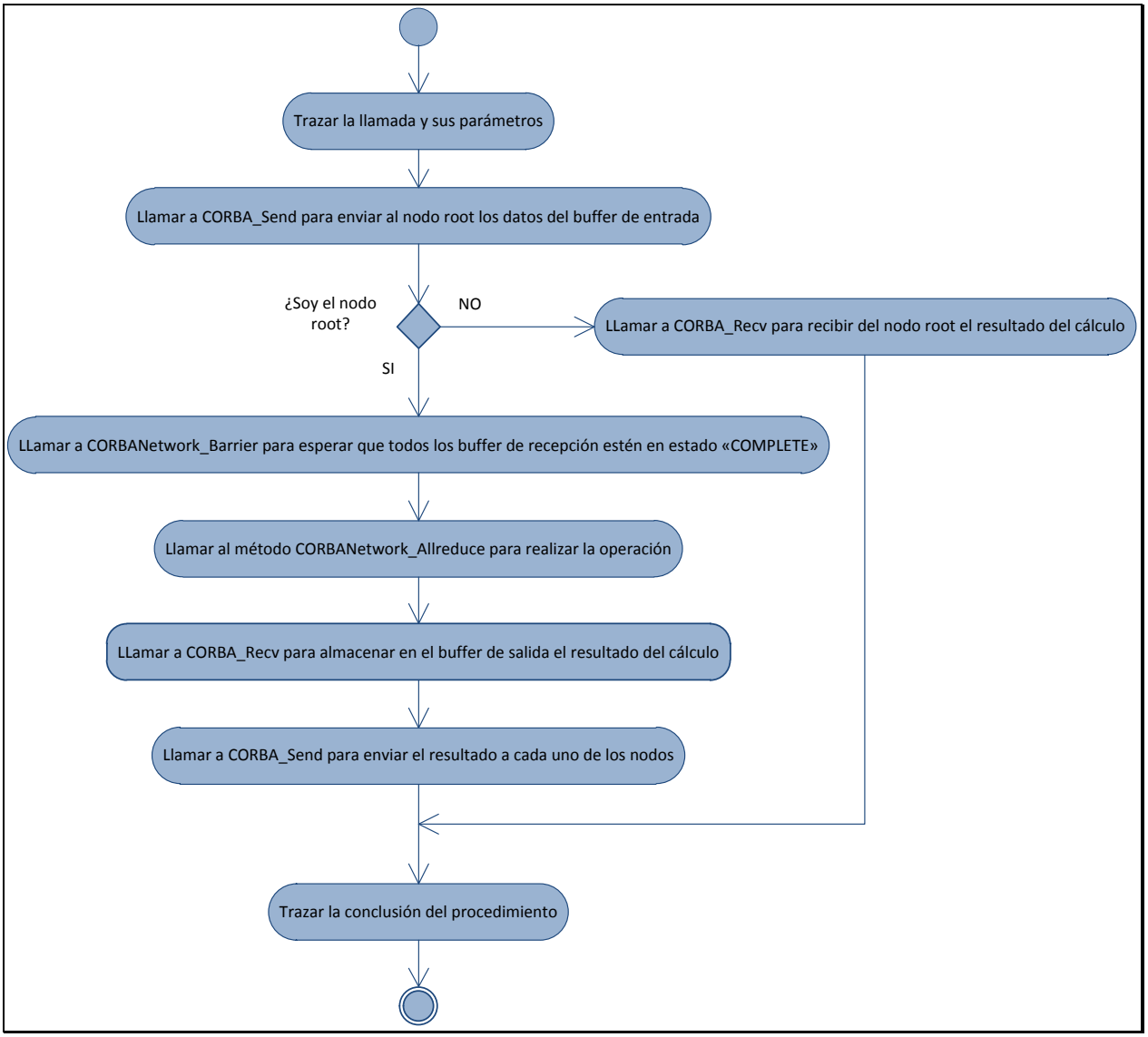


Diagrama 13: Actividad CORBA_Allreduce

Como se observa, se reutilizan las funciones CORBA_Send y CORBA_Recv para las transmisiones de datos, mientras que el método CORBANetwork_Allreduce solo realiza la operación correspondiente. En nuestro caso, la suma de los buffers dejando el resultado en el del nodo root.

Un método importante en esta función es el método CORBANetwork_Barrier, que realiza una espera para que todos los nodos hijos hayan enviado los datos al root.
Su funcionamiento es el siguiente:

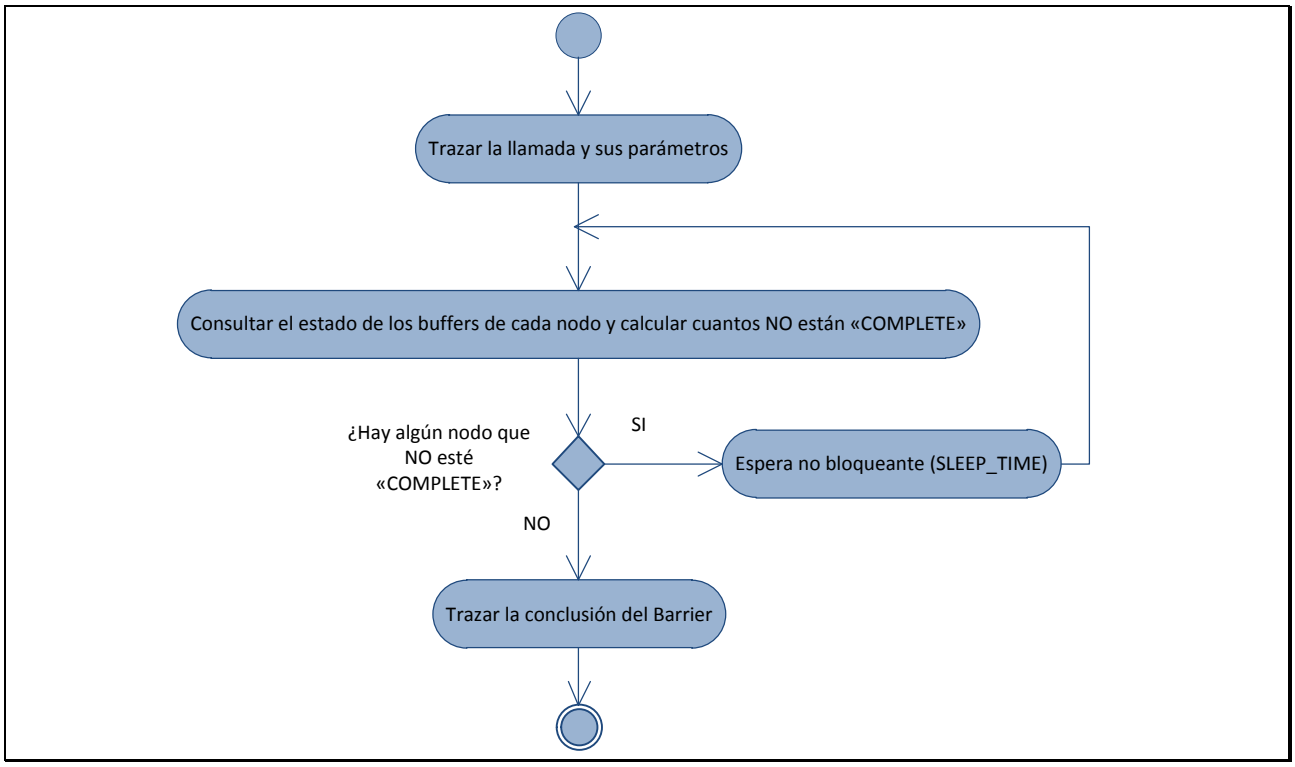


Diagrama 14: Actividad CORBANetwork_Barrier

3.4. Parámetros de configuración

Con este proyecto, hemos desarrollado una versión de GROMACS compatible con múltiples plataformas.
Cada plataforma tiene sus propias características (sistema operativo, número de procesadores/nodos, velocidad del procesamiento, memoria, etc.). Es por ello que hemos decidido incluir directivas para el precompilador que nos permiten configurar determinados parámetros con el fin de optimizar el rendimiento en la ejecución de simulaciones.

Estos parámetros se encuentran en el fichero "*corbanetworkpfc.h*" y son los siguientes:

PARÁMETRO	DESCRIPCIÓN
`#define TRAZASGROMACS`	Si está definida, imprime trazas en todas las funciones GROMACS de comunicaciones.
`#define TRAZASCORBAMPI`	Si está definida, imprime trazas en todas las funciones CORBA para MPI
`#define TRAZASCORBAMPIBUFFERS`	Si está definida y activadas las trazas CORBA para MPI, imprime también el contenido de los buffers
`#define MAX_NUMNODES 32`	Número máximo de nodos permitidos para optimizar el consumo de memoria.
`#define MIN_ESPERA_NODO 100`	Tiempo mínimo de espera a que un nodo arranque (milisegundos)
`#define MAX_ESPERA_NODO 1000000`	Tiempo máximo de espera a que un nodo arranque (milisegundos)
`#define POLLING_INTEVAL 100`	Intervalo de espera para un nuevo intento (milisegundos)
`#define SLEEP_TIME 1`	Tiempo a dormir en procesos de espera (milisegundos)
`#define SIGACK SIGUSR2`	Señal a lanzar cuando se complete una lectura o escritura de un buffer. No usar la señal SIGUSR1 porque la captura GROMACS.

Tabla 2: Parámetros de configuración

3.5. Optimizaciones realizadas

El desarrollo e implementación del proyecto ha pasado por varias etapas. En un primer momento, nos centramos en la localización de las llamadas de envío y recepción de datos (Send, Recv) y su sustitución por llamadas CORBA de forma síncrona.
Completada esta etapa, concluimos que la implementación sería posible pero que tendríamos que completar el desarrollo con el resto de funciones utilizadas y por último realizar optimizaciones en el código con el fin de mejorar los tiempos de ejecución de las simulaciones.

Las optimizaciones realizadas han sido:

- Disminución de los tiempos de espera.
- Mecanismo de reconocimiento de envío/recepción.

Estas optimizaciones han supuesto mejoras en el rendimiento, por lo que se ha implementado una segunda versión de la librería CORBANetwork cuyos resultados frente a la versión inicial se aprecia en los primeros experimentos realizados.

3.5.1 Disminución de los tiempos de espera

En la versión inicial del desarrollo, la implementación de CORBA_Send y CORBA_Recv era totalmente síncrona, es decir, el emisor enviaba los datos al receptor y no devolvía el control hasta completado el envío, y el receptor igualmente se quedaba bloqueado hasta completar la recepción.
En la versión final del desarrollo, la librería lanza un hilo específico para las comunicaciones CORBA. Además, durante las transmisiones de datos, cuando el buffer del destinatario o del receptor no está en disposición de uso, se llama a la función "*msleep*" que realiza una espera no bloqueante, es decir, que cede el control a otros procesos/hilos pudiendo ser interrumpida por señales.

A continuación se muestra un ejemplo de su uso en el método GET:

```
        // Si el buffer del emisor (idSender) no está lleno, esperamos a que lo esté:
        while (servant->bufferState[idSender] != STATE_COMPLETE)
              {
              #ifdef TRAZASCORBA
              tskel = time(NULL); tptrskel = localtime(&tskel);
              fprintf (stdout, "P%d (%2d:%2d:%2d): ", servant->nodeId,
                    tptrskel->tm_hour, tptrskel->tm_min, tptrskel->tm_sec);
              fprintf (stdout, "getInt: Proceso ocupado, nodeState=%d,
                    bufferState=%d\n", servant->nodeState,
                    servant->bufferState[idSender]);
              fflush (stdout);
              #endif

              msleep(SLEEP_TIME);
              }
```

3.5.2 Mecanismo de reconocimiento de envío/recepción

Con la optimización anterior, cuando un nodo envía datos y el receptor está ocupado o viceversa, el hilo de comunicaciones se "duerme" por el tiempo definido en la constante "*SLEEP_TIME*" que es parametrizable para optimizar los rendimientos en función de la plataforma de ejecución.

No obstante, hemos implementado otra mejora que consiste en lanzar una señal de ACK en el nodo destinatario en el momento que se completa un envío (CORBANetwork_SET):

```
        //El envío se ha completado => estado = COMPLETE:
        retval = servant->bufferFloat[idSender]->_length;
        servant->bufferState[idSender] = STATE_COMPLETE;

        // Lanzamos señal ACK para despertar a posibles procesos en espera:
        kill(getpid(),SIGACK);
```

y en el nodo emisor cuando se completa la recepción y por tanto el buffer ya está disponible para futuras comunicaciones (CORBANetwork_GET):

```
        // Lectura completada => estado = READY:
        CORBA_free (servant->bufferFloat[idSender]);
        servant->bufferState[idSender] = STATE_READY;

        // Lanzamos señal ACK para despertar a posibles procesos en espera:
         kill(getpid(),SIGACK);
```

La señal utilizada como ACK es parametrizable, en nuestra implementación se ha utilizado SIGUSR2, que no es una señal no utilizada por el sistema operativo ni por GROMACS.
Inicialmente utilizamos la señal SIGUSR1 pero se tuvo de descartar porque esta señal ya era utilizada durante las simulaciones en la versión original de GROMACS.

Capítulo 4

Estudio comparativo de rendimiento

4.1. Introducción

Para realizar un estudio comparativo del rendimiento de GROMACS utilizando la nueva librería sobre CORBA frente a la actual implementación sobre MPI, hemos utilizado dos plataformas:

A) Equipo local con doble núcleo:

- Procesador: Core Duo T5200 (doble núcleo)
- Frecuencia: 1.6 GHz
- RAM: 2.0 GB
- S.O: Linux Ubuntu 9.04 32 bits, kernel 2.6.28-19

B) Cluster Kasukabe del laboratorio de informática:

- 16 nodos con las siguientes características:
  - Procesador: AMD Athlon XP 1700+
  - Frecuencia: 1.1 GHz
  - RAM: 1.5 GB
  - S.O.: Linux Debian Lenny 32 bits, kernel 2.6.26-1-686
- Nodos interconectados en una red de área local con las siguientes características:
  - Interfase de red: Ethernet de 1 Gbps
  - Switch de 24 bocas a 1 Gbps
- Un gestor de trabajos (kasukabe.lab.inf.uc3m.es):
  - Procesador: Pentium III Dual
  - Frecuencia: 1.0 GHz
  - RAM: 2.0 GB
  - S.O.: Linux Debian Lenny 32 bits, kernel 2.6.26-1-686

4.1.1 Equipo local con doble núcleo

La instalación en el equipo local ha requerido los siguientes pasos:

1) Instalación de la librería FFTW, requerida por GROMACS para cálculos de la transformada de Fourier:

   - Crear la carpeta: /home/javier/fftw
   - Descargar el paquete con los fuentes de FFTW-3.1.2
   - Ejecutar:

     ./configure --prefix=/home/javier/fftw --enable-float --enable-threads
     make
     make install
     export CPPFLAGS=-I/home/javier/fftw/include
     export LDFLAGS=-L/home/javier/fftw/lib

2) Instalación de la implementación MPICH de MPI y servidor ssh desde el paquete binario:

   sudo apt-get install mpich-bin
   sudo apt-get install libmpich1.0-dev
   sudo apt-get install openssh-server

3) Instalación de GROMACS.

   - Crear la carpeta: /home/javier/gromacs
   - Descargar el paquete con los fuentes de GROMACS
   - Ejecutar:

     ./configure --enable-mpi --prefix=/home/javier/gromacs
     make
     make install
     export PATH=/home/javier/gromacs/bin:$PATH

4) Instalación de la implementación ORBit2 de CORBA desde el paquete binario:

   sudo apt-get intall orbit2
   sudo apt-get intall liborbit2-dev

4.1.2 Cluster Kasukabe del laboratorio de informática.

El cluster de ordenadores Kasukabe está instalado en el laboratorio de informática de la Universidad Carlos III de Madrid.
Está compuesto por 16 máquinas con sistema operativo Linux Debian gestionadas por un el gestor de trabajos PBS para soportar la carga de múltiples usuarios. Este gestor es el encargado de enviar trabajos, monitorizar y eliminar los trabajos entre los nodos que componen el cluster.

Para enviar un trabajo al cluster es necesario especificar sus características en un script:

```
#PBS -N gromacs2
#PBS -q long
#PBS -o salida2
#PBS -e error2
#PBS -l nodes=2
#PBS -l walltime=00:35:00

cd $PBS_O_WORKDIR
echo "Contenido de $PBS_NODEFILE: "
cat $PBS_NODEFILE

rm *.log
rm *.ref
mpirun.mpich -np 2 -machinefile $PBS_NODEFILE
/datos_nfs/pfc_aliseda/gromacs/bin/mdrun -v -s pr -e pr -o pr -c
/datos_nfs/pfc_aliseda/speptide/after_pr -g
/datos_nfs/pfc_aliseda/speptide/prlog >
/datos_nfs/pfc_aliseda/speptide/misalida2
exit
```

Con este script se le indican los siguientes parámetros:

SENTENCIA	DESCRIPCIÓN
#PBS -N gromacs2	Nombre del trabajo. Por defecto es el nombre del fichero.
#PBS -q long	Cola de trabajos a utilizar.
#PBS -o salida2	Localización del fichero que contendrá la salida estándar (stdout).
#PBS -e error2	Localización del fichero que contendrá la salida estándar de error (stderr).
#PBS -l nodes=2	Número de nodos que necesitamos. Si los nodos tuvieran dos procesadores (p.e.) se indicaría con "nodes=2:ppn=2".
#PBS -l walltime=00:35:00	Tiempo máximo de ejecución que requiere el trabajo. Pasado ese tiempo el trabajo es eliminado automáticamente de la cola.

Tabla 3: Pámetros trabajos gestor de colas

Tras estos parámetros, se incluye el código del trabajo a realizar.

El gestor dispone de los siguientes comandos:

COMANDO	DESCRIPCIÓN
qsub <script.qsub>	Lanza el trabajo del script que se indica.
qstat -a	Muestra el estado actual de los trabajos lanzados.
qdel <job.ID>	Elimina el trabajo indicado.
qhold <job.ID>	Detiene temporalmente la ejecución del trabajo indicado.

Tabla 4: Comandos gestor de colas

Actualmente el cluster dispone de tres colas:

COLA	CARACTERÍSTICAS
short	Nodos disponibles: solo 1. Max. walltime: 5 min.
normal	Nodos disponibles: 2-8. Max. walltime: 10 min.
long	Nodos disponibles: 16. Max. walltime: 30 min.

Tabla 5: Colas del gestor Kasukabe

Las máquinas que componen el cluster están conectadas en red local y están gestionadas por el gestor de trabajos (kasukabe.lab.inf.uc3m.es), según el diagrama siguiente:

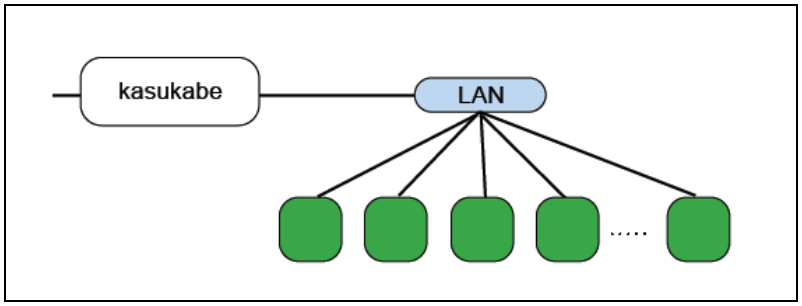


Diagrama 15: Cluster Kasukabe

El equipo kasukabe es el único equipo que está accesible para los usuarios a través de *ssh* para poder lanzar trabajos y monitorizar su ejecución.

4.2. Estudio de la molécula Speptide

4.2.1 Generación de la molécula

La molécula Speptide forma parte de una encima digestiva segregada por el páncreas.
Ha sido muy estudiada experimental y teóricamente por su marcada forma helicoidal en un tamaño pequeño.

Todos los ficheros necesarios para la simulación se distribuyen, a modo de ejemplo, dentro del paquete GROMACS, dentro de la carpeta:

share/tutor/speptide

A continuación se muestra su representación tridimensional obtenida con el programa jmol (http://jmol.sourceforge.net/index.es.html):

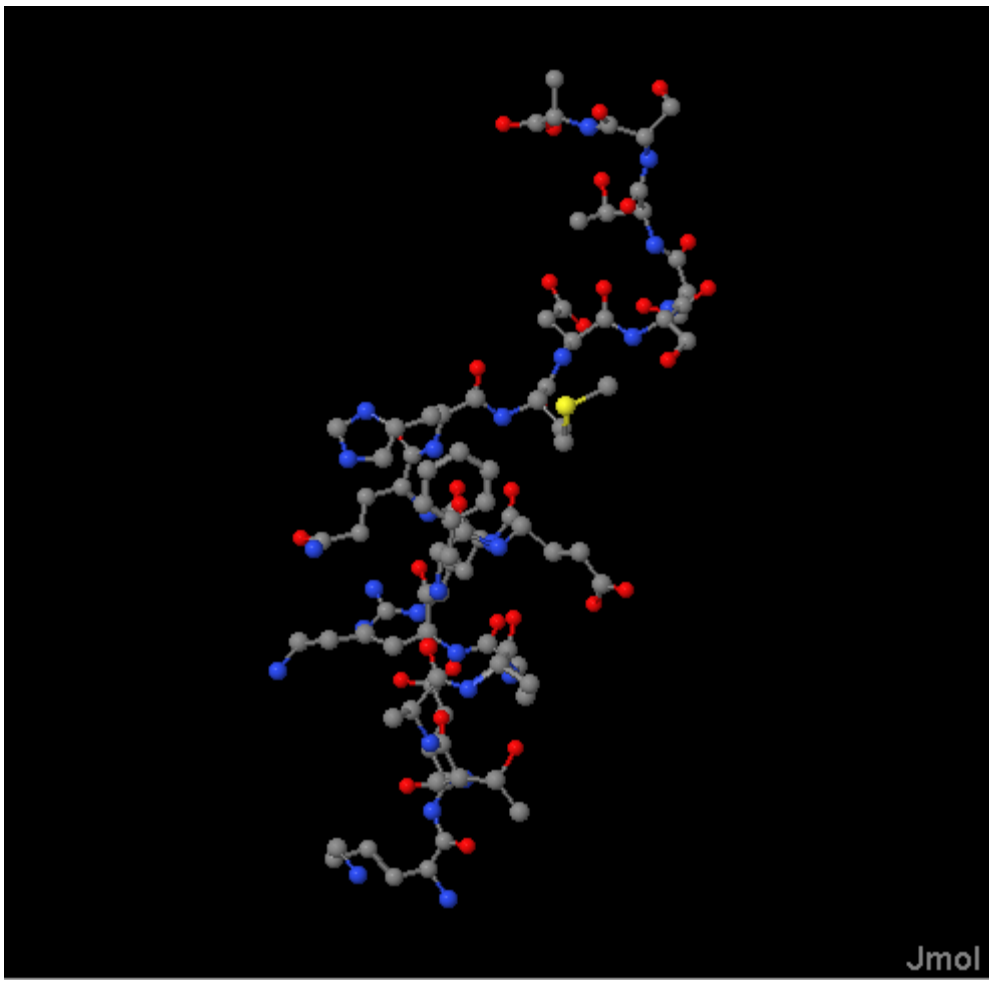


Figura 2: Simulación molécula Speptide

Los pasos utilizados para la preparación de la simulación son los siguientes:

1) *pdb2gmx -f speptide.pdb -p speptide.top -o speptide.gro* [usar opción 0]
2) *editconf -f speptide -o -d 0.5 -c*
3) *genbox -cp out -cs -p speptide -o b4em*
4) *make_ndx -f b4em* [usar opción q]
5) *grompp -v -f em -c b4em -o em -p speptide*
6) *mdrun -v -s em -o em -c after_em -g emlog*
7) *grompp -f pr -o pr -c after_em -r after_em -p speptide*

Una vez preparados los ficheros, para ejecutar la simulación completa de forma secuencial, hemos ejecutado el siguiente comando:

mdrun -v -s pr -e pr -o pr -c after_pr -g prlog

Y para ejecutarlo en paralelo, con MPI y 2 procesos:

mpirun -np 2 mdrun -v -s pr -e pr -o pr -c after_pr -g prlog

La ejecución de la simulación concluye con numerosos datos sobre rendimiento:

R E A L C Y C L E A N D T I M E A C C O U N T I N G

Computing:	Nodes	Number	G-Cycles	Seconds	%
Comm. coord.	2	5001	9.216	8.3	6.5
Neighbor search	2	501	12.183	11.0	8.6
Force	2	5001	82.720	74.8	58.4
Wait + Comm. F	2	5001	20.971	19.0	14.8
Write traj.	2	101	0.425	0.4	0.3
Update	2	5001	2.580	2.3	1.8
Constraints	2	5001	5.131	4.6	3.6
Comm. energies	2	5001	4.354	3.9	3.1
Rest	2		4.018	3.6	2.8
Total	2		141.597	128.0	100.0

M E G A - F L O P S A C C O U N T I N G

Parallel run - timing based on wallclock.
RF=Reaction-Field FE=Free Energy SCFE=Soft-Core/Free Energy
T=Tabulated W3=SPC/TIP3p W4=TIP4p (single or pairs)
NF=No Forces

Computing:	M-Number	M-Flops	% Flops
LJ	36.523707	1205.282	1.5
Coulomb	114.152884	3082.128	3.8
Coulomb [W3]	6.851060	548.085	0.7
Coulomb + LJ	42.578120	1617.969	2.0
Coulomb + LJ [W3]	19.723669	1794.854	2.2
Coulomb + LJ [W3-W3]	266.406701	65269.642	81.3
Outer nonbonded loop	72.190002	721.900	0.9
1,4 nonbonded interactions	1.580316	142.228	0.2
NS-Pairs	102.532798	2153.189	2.7
Reset In Box	0.466431	1.399	0.0
Shift-X	27.415482	164.493	0.2
CG-CoM	1.373241	4.120	0.0
Sum Forces	13.707741	13.708	0.0
Angles	1.385277	232.727	0.3
Propers	0.510102	116.813	0.1
Impropers	0.430086	89.458	0.1
Pos. Restr.	0.730146	36.507	0.0
Virial	13.977795	251.600	0.3
Update	13.707741	424.940	0.5
Stop-CM	13.707741	137.077	0.2
Calc-Ekin	13.710482	370.183	0.5
Lincs	0.960576	57.635	0.1
Lincs-Mat	16.629972	66.520	0.1
Constraint-V	14.678802	117.430	0.1
Constraint-Vir	13.718226	329.237	0.4
Settle	4.252550	1373.574	1.7
Total		80322.698	100.0

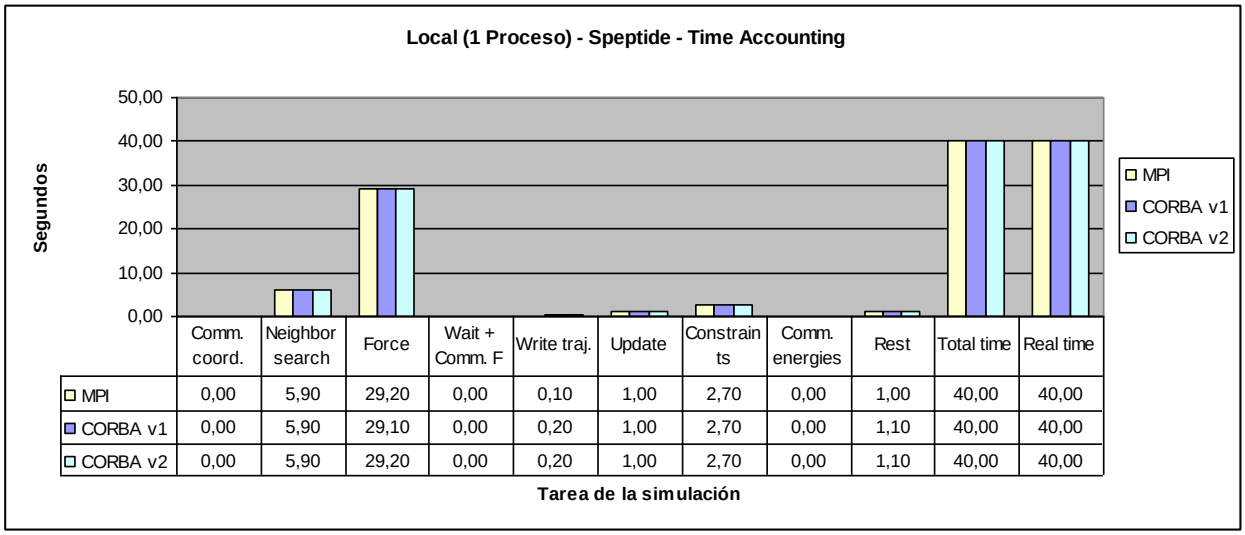
	NODE (s)	Real (s)	(%)
Time:	64.000	64.000	100.0
	1:04		
	(Mnbf/s)	(GFlops)	(ns/day)	(hour/ns)
Performance:	7.597	1.255	13.500	1.778

Tabla 6: Cómputo de tiempos en GROMACS

4.2.2 Comparativa de tiempos en equipo local

A continuación vamos a comparar los datos de rendimiento más significativos entre las simulaciones con MPI y las simulaciones con CORBA tanto en la primera implementación como en la segunda que supuso una optimización de la primera.

En primer lugar analizamos la ejecución en un solo proceso:



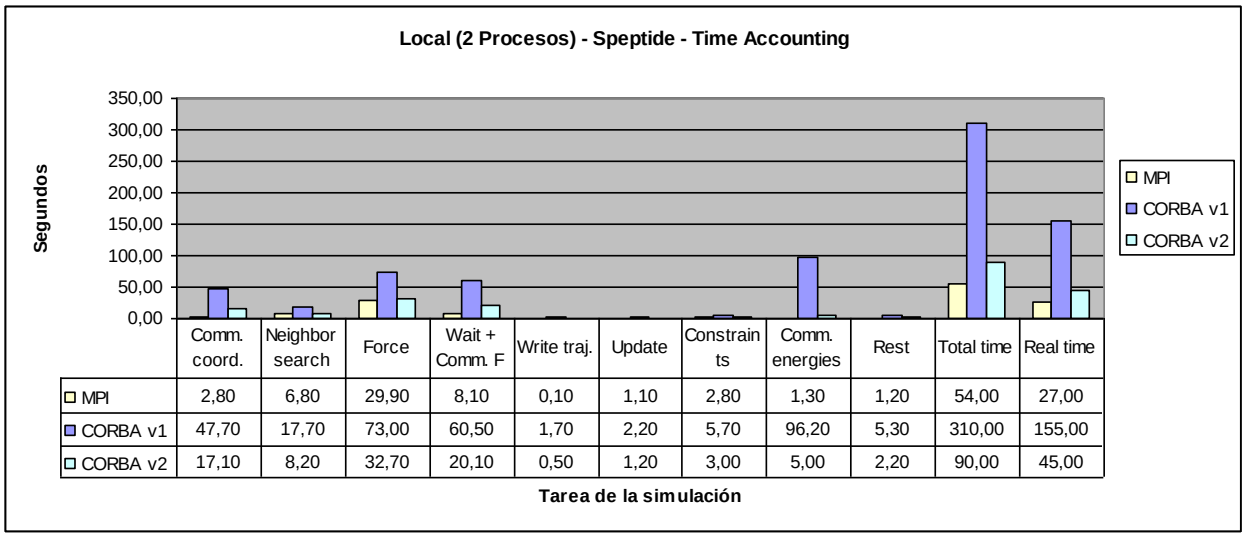
	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
MPI	0,00	5,90	29,20	0,00	0,10	1,00	2,70	0,00	1,00	40,00	40,00
CORBA v1	0,00	5,90	29,10	0,00	0,20	1,00	2,70	0,00	1,10	40,00	40,00
CORBA v2	0,00	5,90	29,20	0,00	0,20	1,00	2,70	0,00	1,10	40,00	40,00

Figura 3: Speptide. Tiempos en local con 1 proceso

Como se observa, el tiempo de ejecución es el mismo en las tres implementaciones, porque realmente no se realiza ninguna de las tareas que implican comunicaciones:

- Comm. coord: Comunicación de coordenadas.
- Wait + Comm. F.: Espera y comunicación de fuerzas.
- Comm. energies: Comunicación de energías.

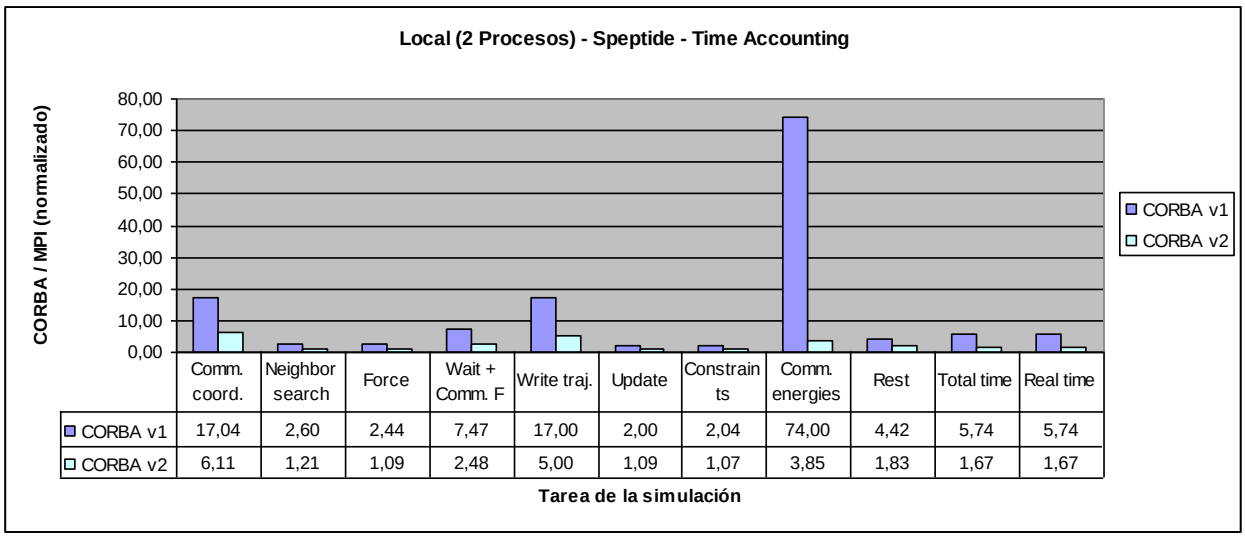
En la simulación con dos procesos obtuvimos:



	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
MPI	2,80	6,80	29,90	8,10	0,10	1,10	2,80	1,30	1,20	54,00	27,00
CORBA v1	47,70	17,70	73,00	60,50	1,70	2,20	5,70	96,20	5,30	310,00	155,00
CORBA v2	17,10	8,20	32,70	20,10	0,50	1,20	3,00	5,00	2,20	90,00	45,00

Figura 4: Speptide. Tiempos en local con 2 procesos

Si normalizamos los valores anteriores con respecto a MPI obtenemos la siguiente gráfica:

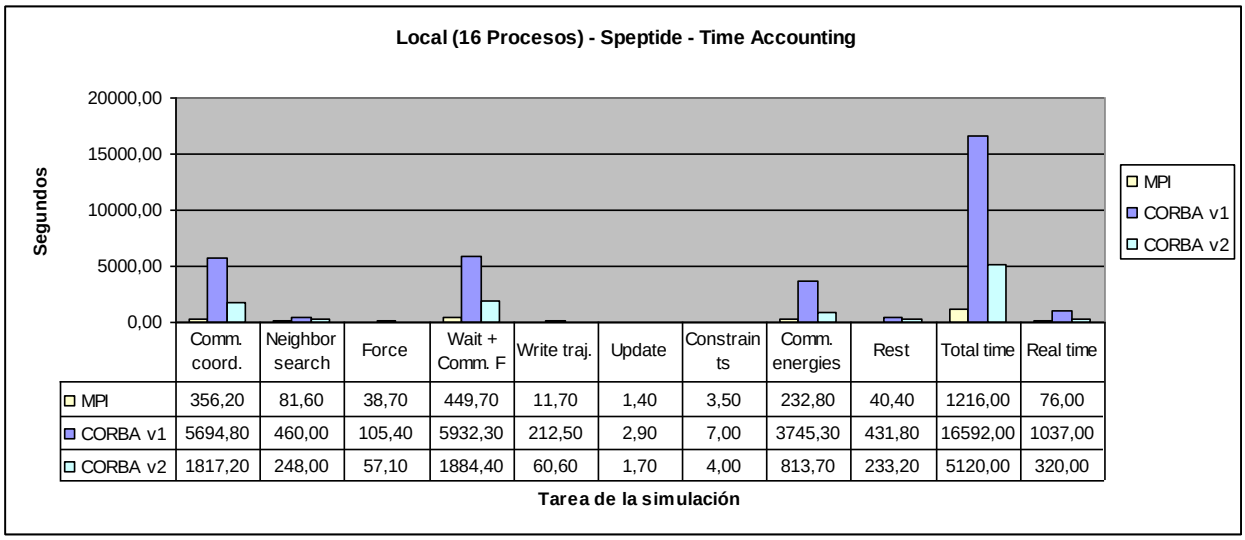


	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
CORBA v1	17,04	2,60	2,44	7,47	17,00	2,00	2,04	74,00	4,42	5,74	5,74
CORBA v2	6,11	1,21	1,09	2,48	5,00	1,09	1,07	3,85	1,83	1,67	1,67

Figura 5: Speptide. Tiempos en local 2 procesos. Normalizado

En este caso se observa que el tiempo destinado a las comunicaciones es importante, pero a pesar de ello, en el caso de MPI el tiempo real de la simulación disminuye a 27, pero en el caso de CORBA aumenta a 155 para la v1 y a 45 para la v2.
Se confirma un incremento significativo de tiempos, aunque también se observa una gran mejoría entre la versión CORBA v2 frente a la v1.

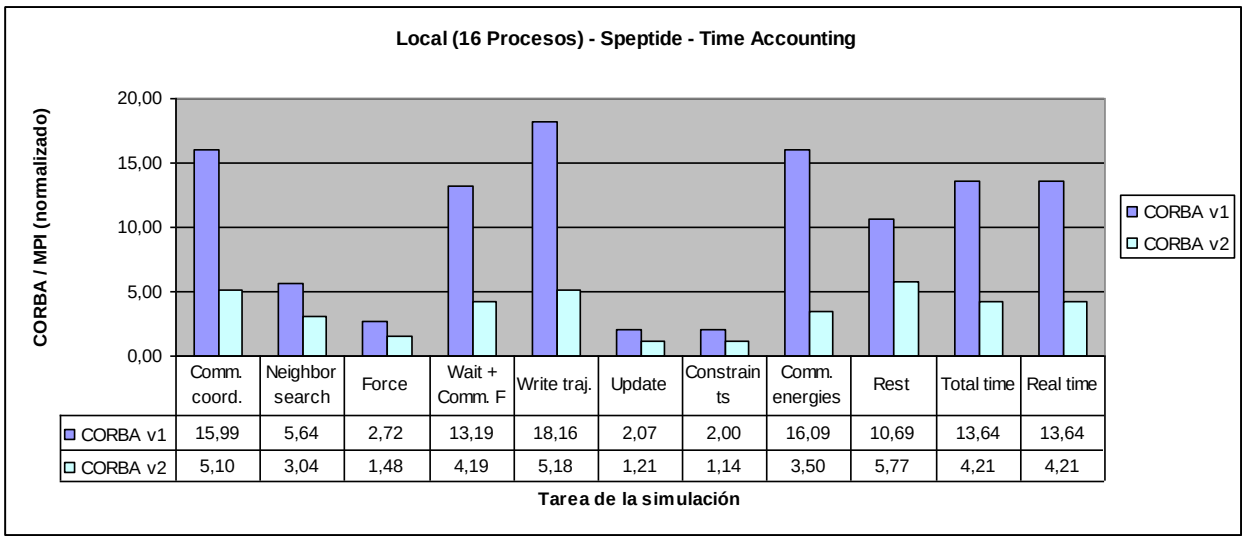
Si seguimos ampliando procesos, en el caso de 16 obtenemos los siguientes datos:



	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
MPI	356,20	81,60	38,70	449,70	11,70	1,40	3,50	232,80	40,40	1216,00	76,00
CORBA v1	5694,80	460,00	105,40	5932,30	212,50	2,90	7,00	3745,30	431,80	16592,00	1037,00
CORBA v2	1817,20	248,00	57,10	1884,40	60,60	1,70	4,00	813,70	233,20	5120,00	320,00

Figura 6: Speptide. Tiempos en local con 16 procesos

Si normalizamos los valores anteriores con respecto a MPI obtenemos la siguiente gráfica:



	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
CORBA v1	15,99	5,64	2,72	13,19	18,16	2,07	2,00	16,09	10,69	13,64	13,64
CORBA v2	5,10	3,04	1,48	4,19	5,18	1,21	1,14	3,50	5,77	4,21	4,21

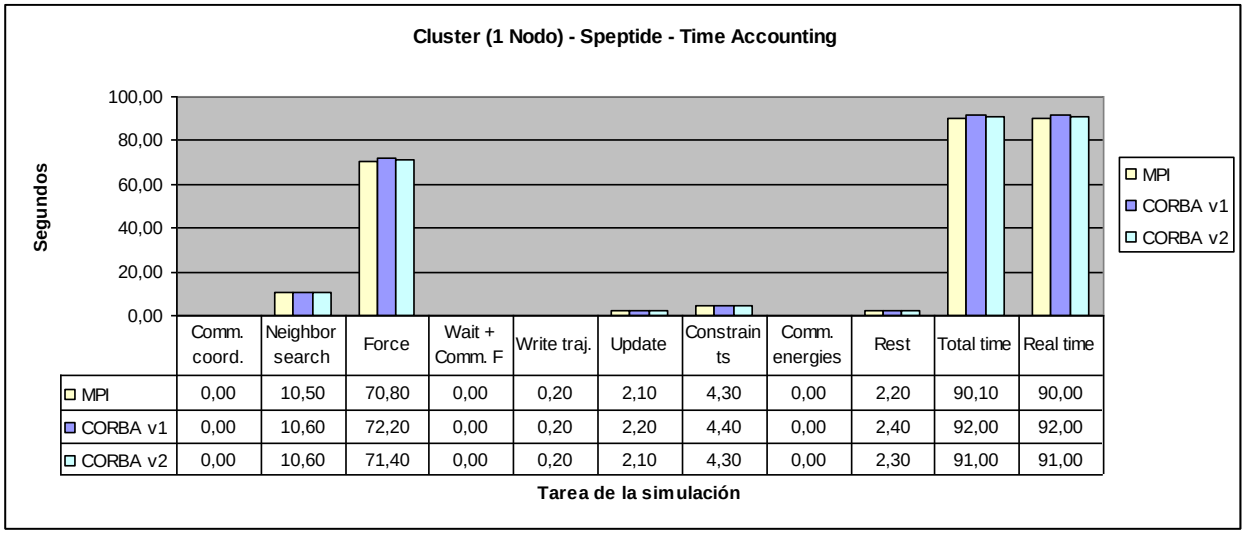
Figura 7: Speptide. Tiempos en local con 16 nodos. Normalizado

En ese caso observamos, que aunque usemos muchos nodos, al tratarse de un solo equipo (aunque con dos núcleos), los tiempos se incrementan respecto a MPI.
Se confirma que la incluso con la versión CORBA v2 los tiempos son más de 4 veces superiores a los de MPI.

4.2.3 Comparativa de tiempos en cluster Kasukabe

En una segunda fase de experimentación, decidimos ejecutar la simulación en el cluster Kasukabe.

En primer lugar analizamos la ejecución con un solo nodo:

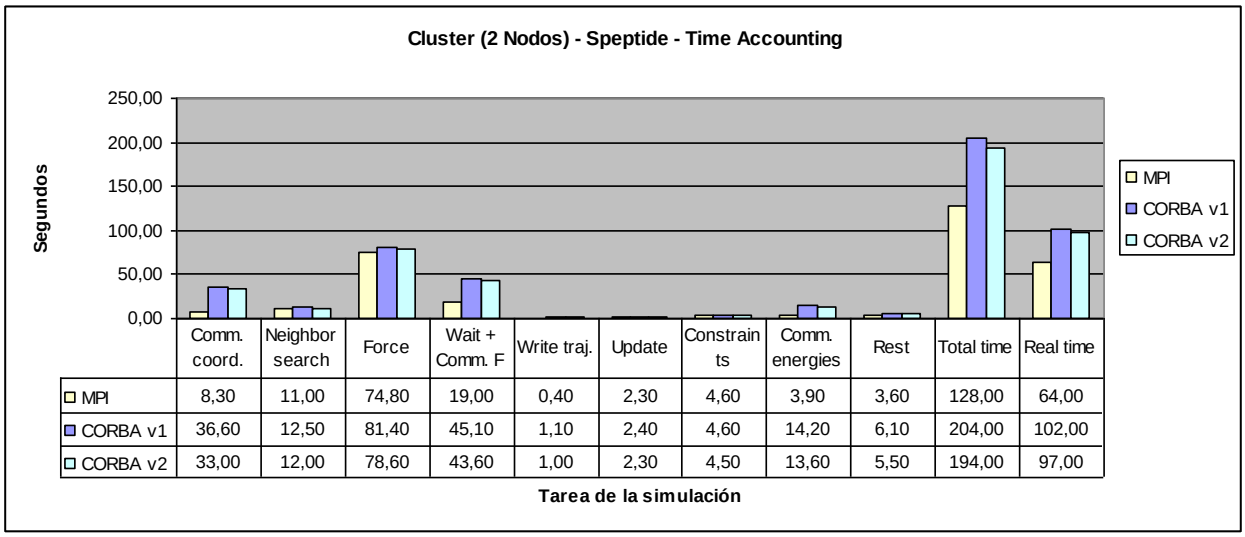


	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
MPI	0,00	10,50	70,80	0,00	0,20	2,10	4,30	0,00	2,20	90,10	90,00
CORBA v1	0,00	10,60	72,20	0,00	0,20	2,20	4,40	0,00	2,40	92,00	92,00
CORBA v2	0,00	10,60	71,40	0,00	0,20	2,10	4,30	0,00	2,30	91,00	91,00

Figura 8: Speptide. Tiempos en cluster usando 1 nodo

Como se puede observar, el tiempo de ejecución es prácticamente el mismo en las tres implementaciones, porque realmente no se realiza ninguna tarea de comunicaciones, pero se observa que los tiempos han aumentado frente a la simulación en el equipo local. Esto es debido a que las máquinas del cluster son menos potentes.

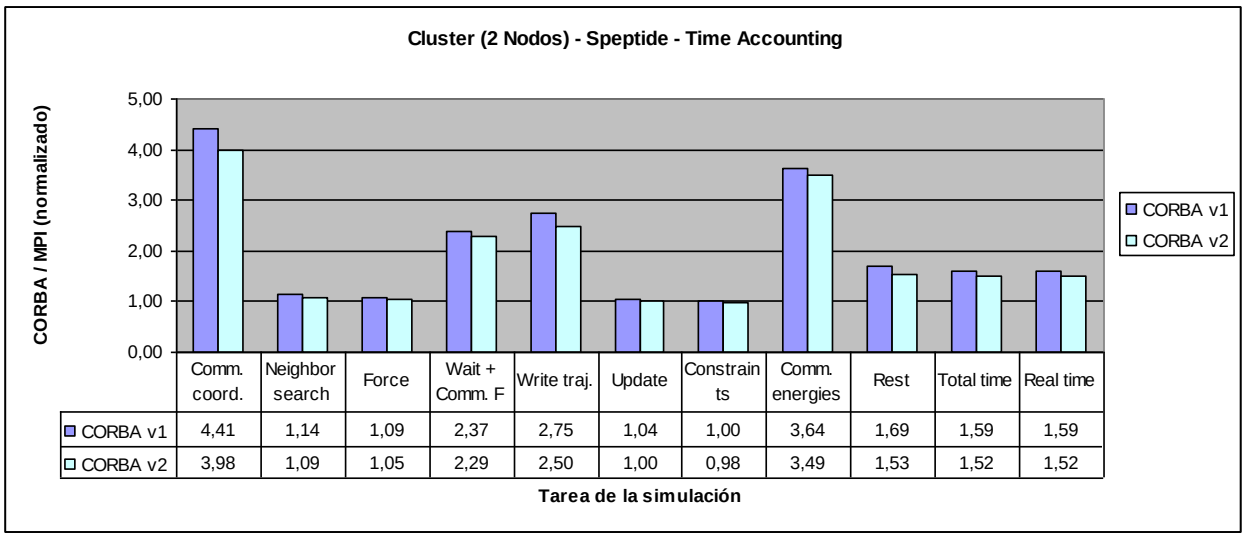
En la simulación con dos nodos obtuvimos:



	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
MPI	8,30	11,00	74,80	19,00	0,40	2,30	4,60	3,90	3,60	128,00	64,00
CORBA v1	36,60	12,50	81,40	45,10	1,10	2,40	4,60	14,20	6,10	204,00	102,00
CORBA v2	33,00	12,00	78,60	43,60	1,00	2,30	4,50	13,60	5,50	194,00	97,00

Figura 9: Speptide. Tiempos en cluster usando 2 nodos

Si normalizamos los valores anteriores con respecto a MPI obtenemos la siguiente gráfica:

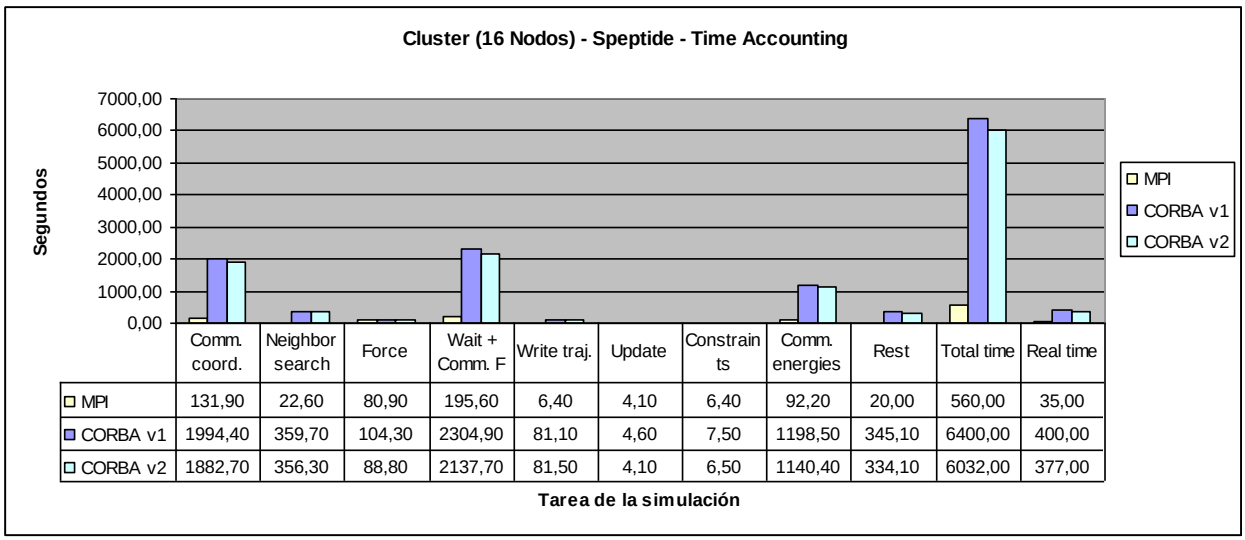


	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
CORBA v1	4,41	1,14	1,09	2,37	2,75	1,04	1,00	3,64	1,69	1,59	1,59
CORBA v2	3,98	1,09	1,05	2,29	2,50	1,00	0,98	3,49	1,53	1,52	1,52

Figura 10: Speptide. Tiempos en cluster usando 2 nodos. Normalizado

En este caso se observa que el tiempo destinado a las comunicaciones es importante, pero a pesar de ello, en el caso de MPI el tiempo real de la simulación disminuye a 64, pero en el caso de CORBA aumenta a 102 para la v1 y a 97 para la v2.
Se aprecia que la tiempo total (Real time) de la versión CORBA es aproximadamente una vez y media la de MPI.
La versión CORBA v2 mejora los tiempos, pero sobre esta plataforma (cluster Kasukabe) la mejora no es tan significativa como en el equipo local.

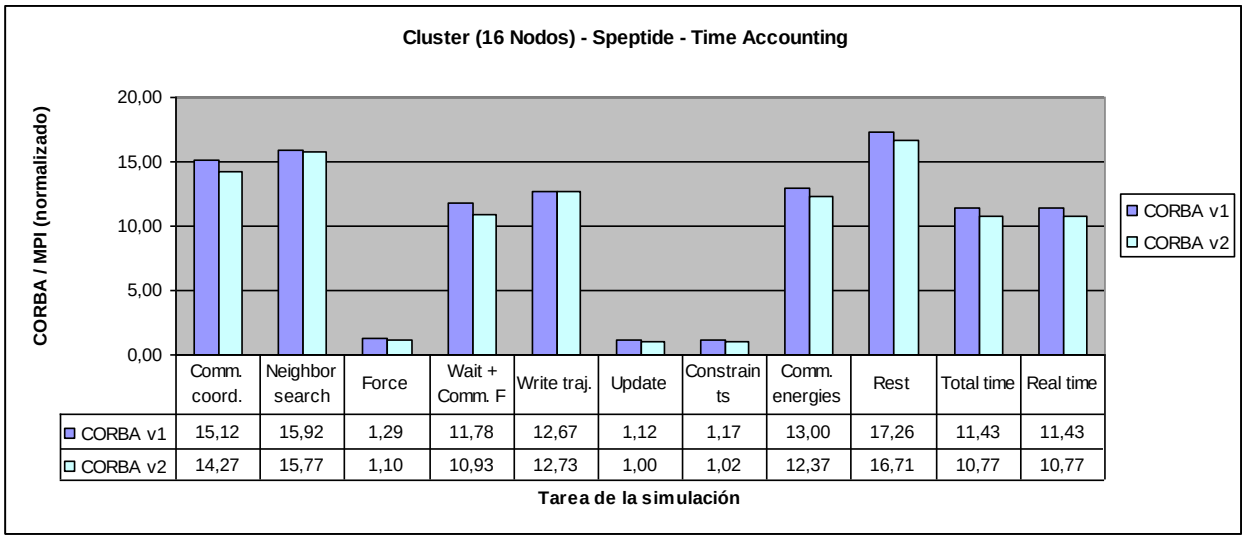
Si seguimos ampliando nodos, en el caso de 16 obtenemos los siguientes datos:



	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
MPI	131,90	22,60	80,90	195,60	6,40	4,10	6,40	92,20	20,00	560,00	35,00
CORBA v1	1994,40	359,70	104,30	2304,90	81,10	4,60	7,50	1198,50	345,10	6400,00	400,00
CORBA v2	1882,70	356,30	88,80	2137,70	81,50	4,10	6,50	1140,40	334,10	6032,00	377,00

Figura 11: Speptide. Tiempos en cluster usando 16 nodos

Si normalizamos los valores anteriores con respecto a MPI obtenemos la siguiente gráfica:



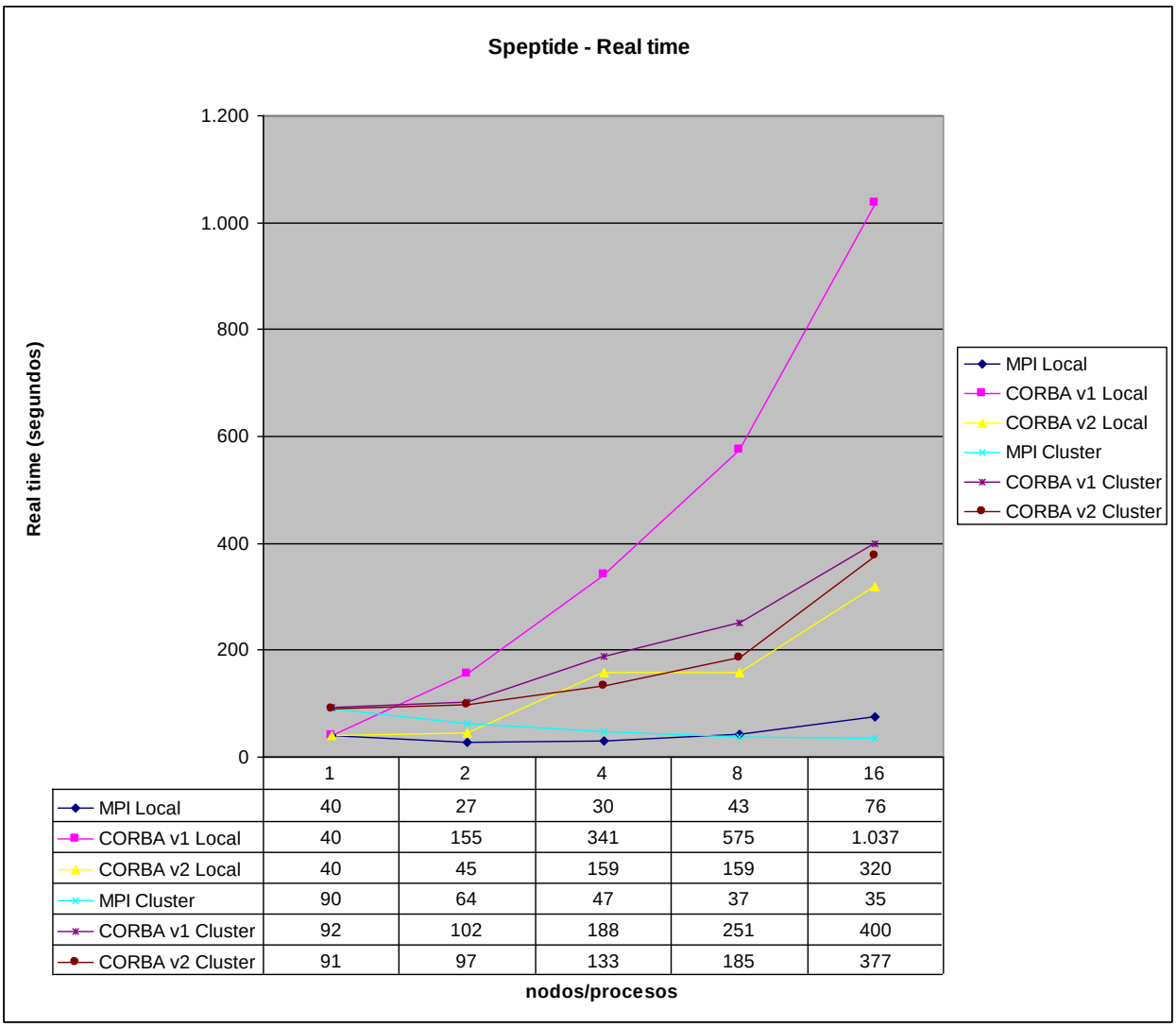
	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
CORBA v1	15,12	15,92	1,29	11,78	12,67	1,12	1,17	13,00	17,26	11,43	11,43
CORBA v2	14,27	15,77	1,10	10,93	12,73	1,00	1,02	12,37	16,71	10,77	10,77

Figura 12: Speptide. Tiempos en cluster usando 16 nodos. Normalizado

En ese caso observamos que usando muchos nodos en cluster, el tiempo real disminuye para la versión MPI a 35 segundos, pero los tiempos de la versión CORBA son más de 10 veces superiores a los de MPI. Este aumento de tiempos de la versión CORBA es superior que en las pruebas en el equipo local, esto responde a que ahora los nodos se deben reconocer en un entorno de red local y hacer la transmisión de datos a través de la LAN, mientras que en local se hacía dentro del mismo equipo.

4.2.4 Resumen comparativa de tiempos

Como resumen, se analiza con la siguiente gráfica el tiempo real de ejecución de la simulación en cada uno de los equipos (local y cluster), para cada una de las implementaciones (MPI, CORBA v1 y CORBA v2) en función del número de procesos/nodos:



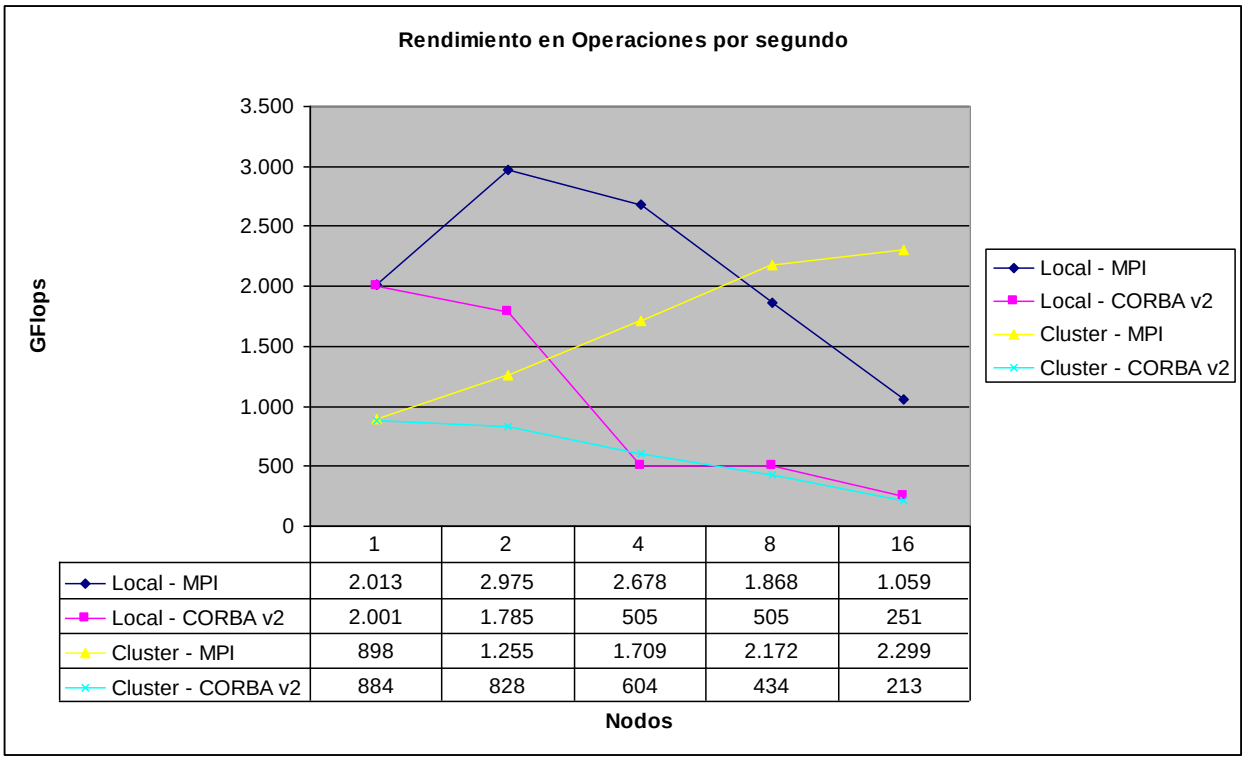
	1	2	4	8	16
MPI Local	40	27	30	43	76
CORBA v1 Local	40	155	341	575	1.037
CORBA v2 Local	40	45	159	159	320
MPI Cluster	90	64	47	37	35
CORBA v1 Cluster	92	102	188	251	400
CORBA v2 Cluster	91	97	133	185	377

Figura 13: Speptide. Comparativa de tiempos

Se puede concluir, que la ejecución de MPI en local disminuye el tiempo para 2 y 4 nodos, pero aumenta a partir de 8.
La ejecución MPI en cluster supone una disminución del tiempo, incluso con 16 nodos, pero el porcentaje de reducción disminuye conforme aumenta el número de nodos.
Por último, la implementación CORBA no consigue disminuir los tiempos en ningún caso.
No obstante, la versión CORBA v2 consigue mejorar los tiempos significativamente frente a la v1.

4.2.5 Comparativa de rendimiento en operaciones por segundo

Desde otra perspectiva, podemos analizar las simulaciones desde el punto de vista del rendimiento, el número de operaciones por segundo que se ha empleado GROMACS en completar la simulación.
Se muestra a continuación la gráfica comparativa, en GFlops, para la molécula Speptide:



	1	2	4	8	16
Local - MPI	2.013	2.975	2.678	1.868	1.059
Local - CORBA v2	2.001	1.785	505	505	251
Cluster - MPI	898	1.255	1.709	2.172	2.299
Cluster - CORBA v2	884	828	604	434	213

Figura 14: Speptide. Comparativa de rendimiento en operaciones por segundo.

De estos datos se extraen las siguientes conclusiones:

- La versión CORBA v2, en ambas plataformas, reduce el número de operaciones por segundo conforme aumenta el número de nodos, debido a que realiza el mismo trabajo pero emplea más tiempo en comunicaciones.
- En el cluster, la versión MPI aumenta progresivamente el número de operaciones por segundo conforme aumenta el número de nodos.
- En el equipo local, la versión MPI, al contar con doble núcleo, con 2 procesos aumenta el número de operaciones, pero a partir de 2 disminuye.

4.2.6 Comparativa de consumo de memoria RAM

Por último, vamos comparar en el equipo local, el consumo de memoria de la versión de GROMACS con CORBA v2 frente a la versión MPI.

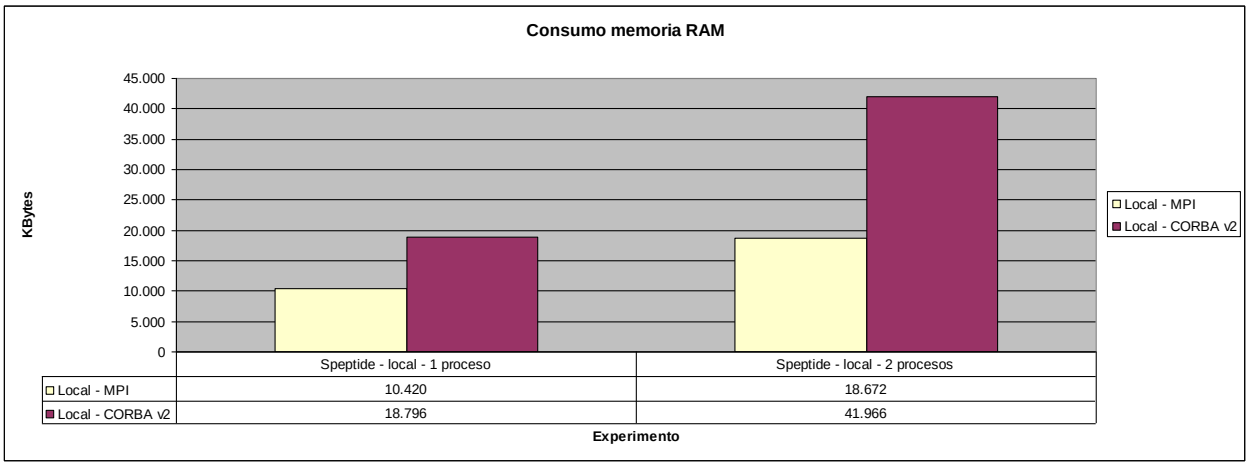


Figura 15: Speptide. Comparativa consumo memoria RAM

Como se observa en la gráfica, la versión CORBA v2 de GROMACS, tanto en la simulación en secuencial (con un solo proceso), como en paralelo (con dos procesos), consume casi 2 veces la memoria de la versión MPI.
Esto es debido a que durante el inicio de la simulación, la versión CORBA v2 debe cargar el middleware de ORBit2 y crear un nuevo objeto de la librería CORBANetwork que se ejecuta en un nuevo hilo independiente, con lo cual se produce un aumento en el consumo de memoria.

Este resultado es extensible a los experimentos realizados en cluster, y por lo tanto, la suma de la memoria RAM consumida con la versión CORBA v2 por cada uno de los nodos implicados en el experimento será en conjunto superior a versión MPI.

4.3. Estudio del virus de la Hepatitis C

4.3.1 Generación del virus

Para hacer simulaciones de mayor envergadura, hemos analizado también una molécula relacionada con el virus la Hepatitis C, obtenida del banco de proteínas (www.rcsb.org) con el código 1R7E. Su fichero de definición (*.pdb*) ocupa 52,4 KB, lo que da una idea de su complejidad frente a la anterior Speptide, con un fichero .pdb de 11,6 KB.

A continuación se muestra una representación tridimensional de esta molécula (obtenida de http://www.rcsb.org/pdb/explore/jmol.do?structureId=1R7E):

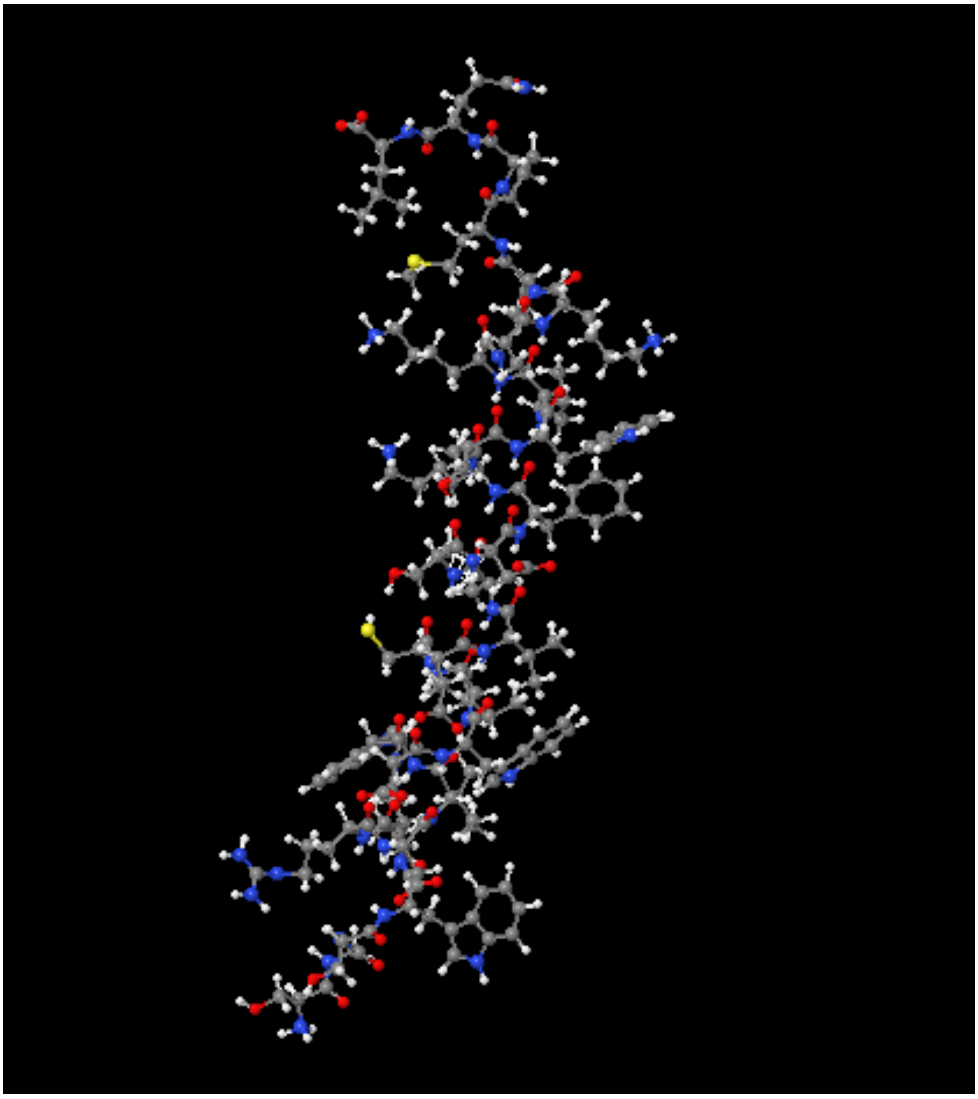


Figura 16: Simulación molécula Hepatitis C

Los pasos utilizados para la generación de la simulación han sido los mismos que para la Speptide, salvo el primero, para el cual se ha utilizado la siguiente variación:

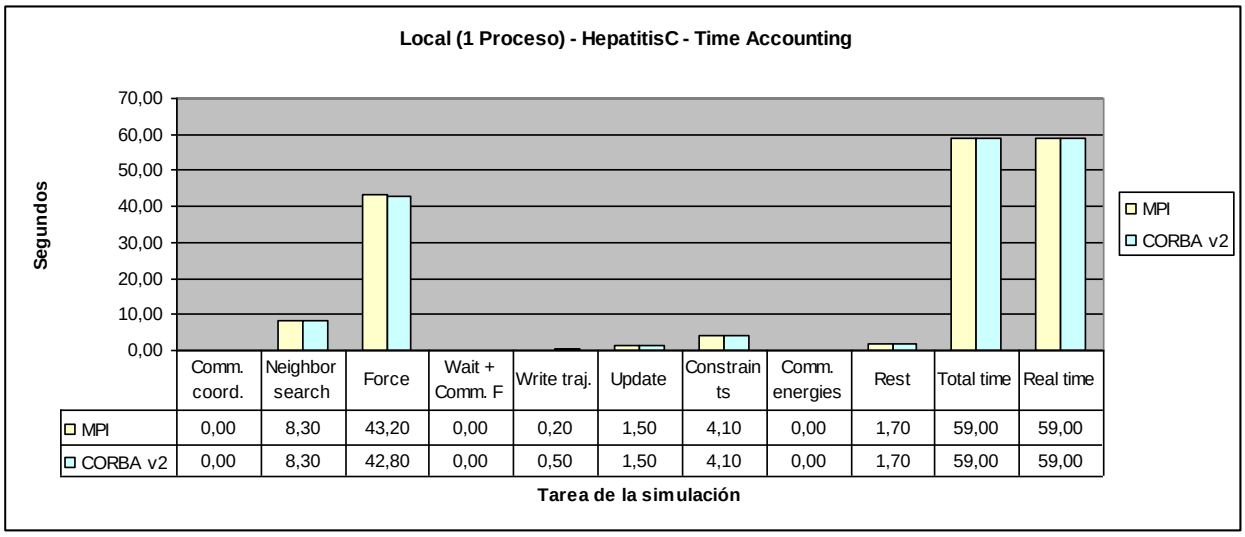
1) *pdb2gmx -ignh -f speptide.pdb -p speptide.top -o speptide.gro*

Hay que notar que se ha renombrado el virus de 17RE.pdb a speptide.pdb para reutilizar los script ya desarrollados para el estudio anterior.

4.3.2 Comparativa de tiempos en equipo local

A continuación vamos a comparar los datos de rendimiento más significativos entre las simulaciones con MPI y las simulaciones con CORBA. En esta ocasión descartamos la implementación CORBA v1 y nos centramos en los datos de rendimiento más significativos entre las simulaciones con MPI y las simulaciones con CORBA v2.

En primer lugar analizamos la ejecución en un solo proceso:

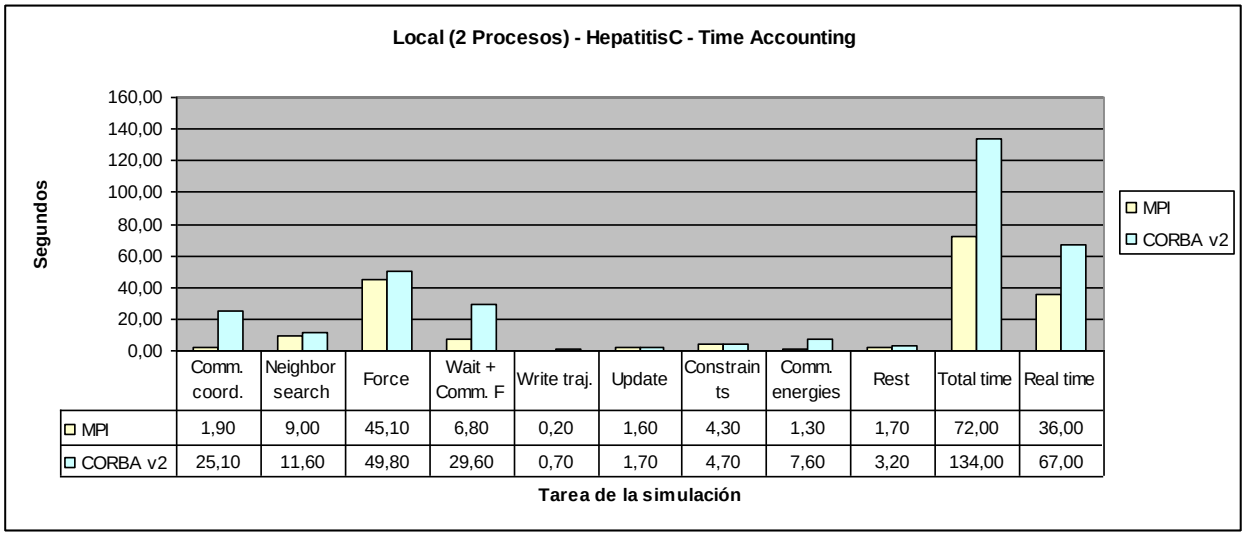


	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
MPI	0,00	8,30	43,20	0,00	0,20	1,50	4,10	0,00	1,70	59,00	59,00
CORBA v2	0,00	8,30	42,80	0,00	0,50	1,50	4,10	0,00	1,70	59,00	59,00

Figura 17: Hepatitis C. Tiempos en local con 1 proceso

Como se observa, el tiempo de ejecución es el mismo en las tres implementaciones, porque realmente no se realiza ninguna comunicación, pero ha aumentado a 59 segundos, frente a los 40 que tardaba la simulación de Speptide.

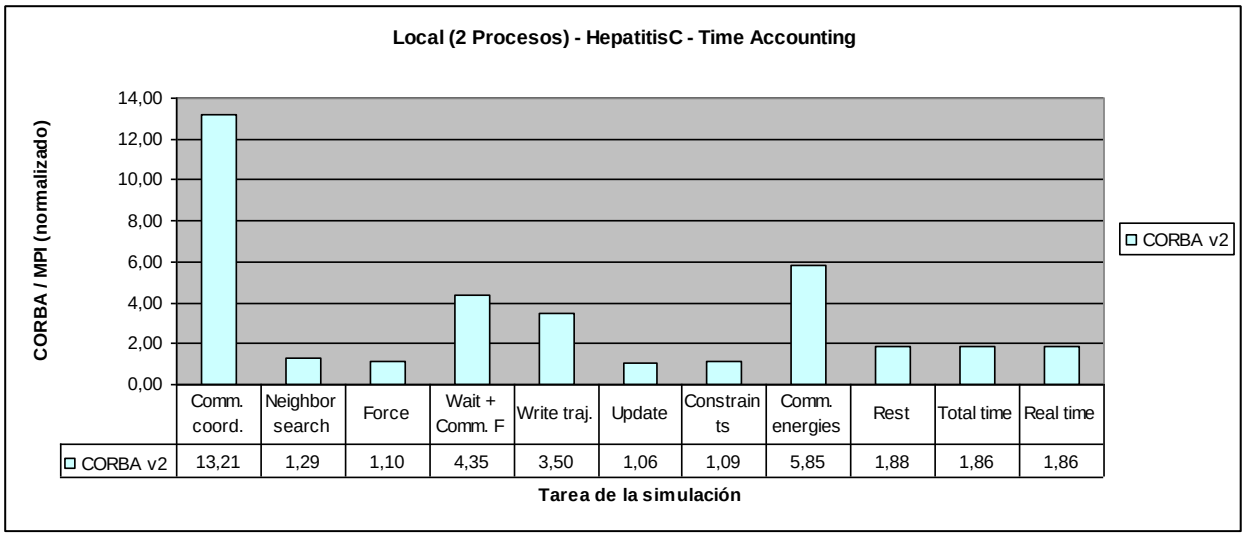
En la simulación con dos procesos obtuvimos:



	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
MPI	1,90	9,00	45,10	6,80	0,20	1,60	4,30	1,30	1,70	72,00	36,00
CORBA v2	25,10	11,60	49,80	29,60	0,70	1,70	4,70	7,60	3,20	134,00	67,00

Figura 18: Hepatitis C. Tiempos en local con 2 procesos

Si normalizamos los valores anteriores con respecto a MPI obtenemos la siguiente gráfica:

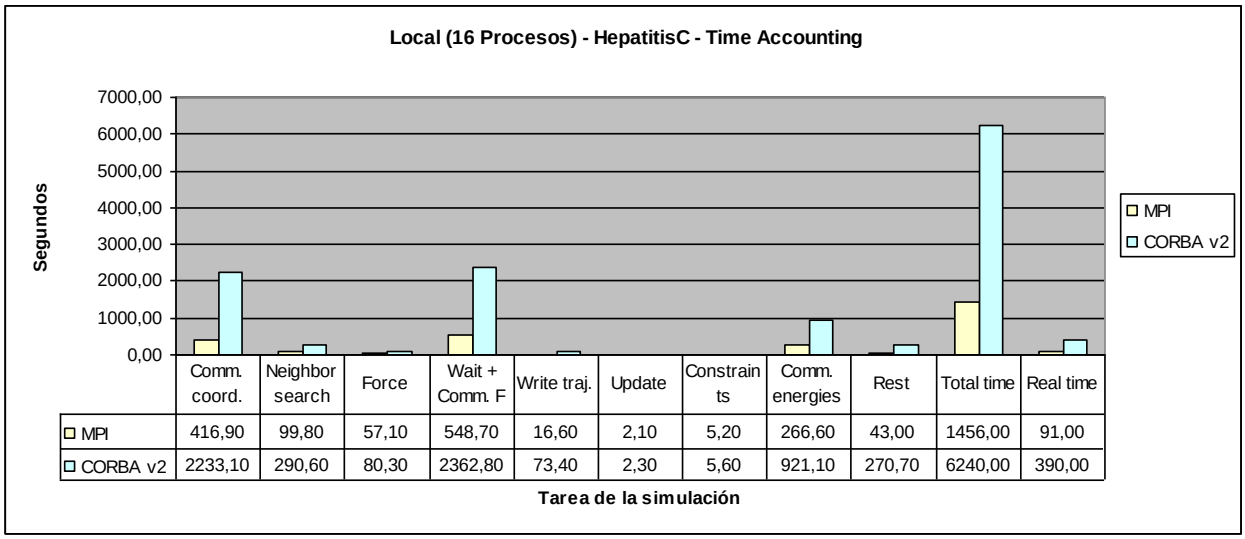


	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
CORBA v2	13,21	1,29	1,10	4,35	3,50	1,06	1,09	5,85	1,88	1,86	1,86

Figura 19: Hepatitis C. Tiempos en local con 2 procesos. Normalizado

En este caso se observa que el tiempo destinado a las comunicaciones es importante, pero a pesar de ello, en el caso de MPI el tiempo real de la simulación disminuye a 36, pero en el caso de CORBA aumenta a 67.
Proporcionalmente, el tiempo real de la versión CORBA v2 es 1,86 veces superior al de MPI.

Si seguimos ampliando procesos, en el caso de 16 obtenemos los siguientes datos:



	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
MPI	416,90	99,80	57,10	548,70	16,60	2,10	5,20	266,60	43,00	1456,00	91,00
CORBA v2	2233,10	290,60	80,30	2362,80	73,40	2,30	5,60	921,10	270,70	6240,00	390,00

Figura 20: Hepatitis C. Tiempos en local con 16 procesos

Si normalizamos los valores anteriores con respecto a MPI obtenemos la siguiente gráfica:

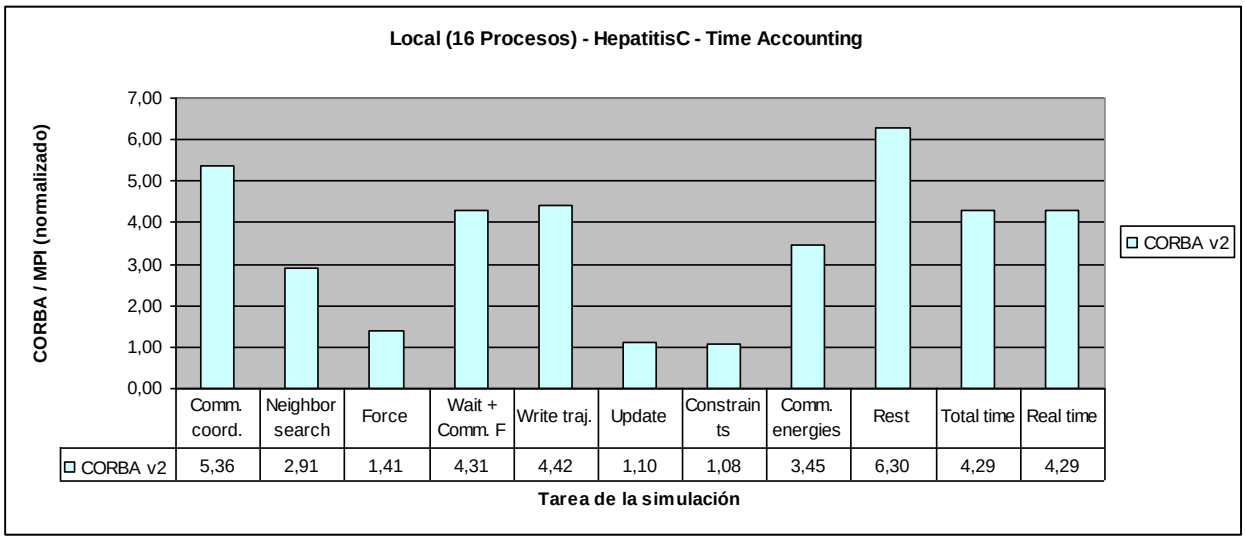


Figura 21: Hepatitis C. Tiempos en local con 16 procesos. Normalizado

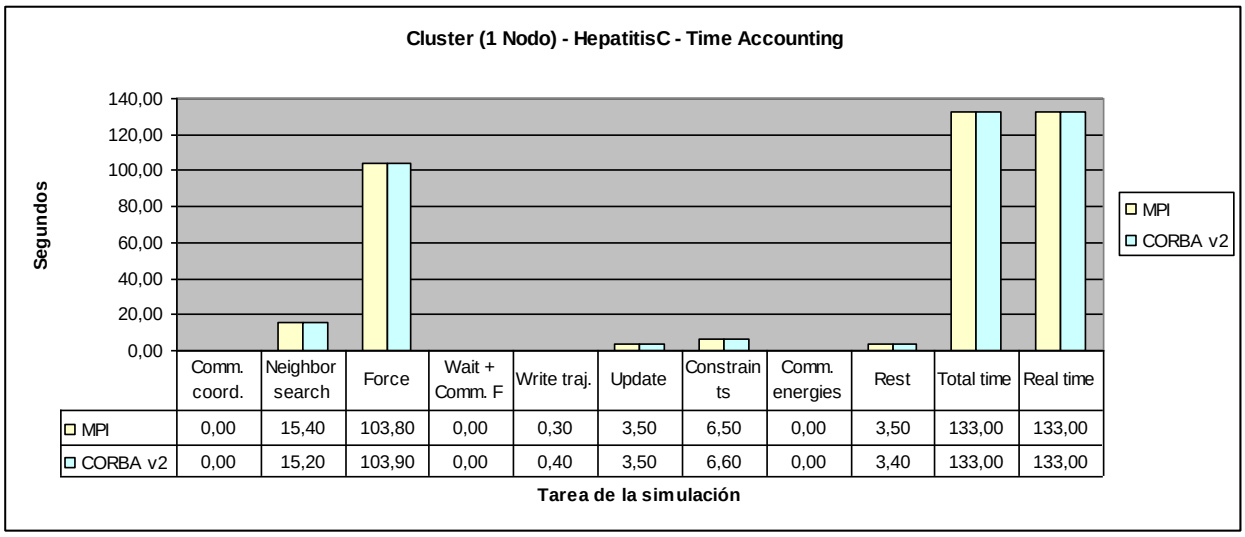
En ese caso observamos, que aunque usemos muchos nodos, al tratase de un solo equipo (aunque con dos núcleos), los tiempos aumentan para la versión MPI y en el caso de CORBA son mucho mayores que los de MPI.
Proporcionalmente, el tiempo real de la versión CORBA v2 es 4,29 veces superior al de MPI.

Se comprueba, por tanto, que al tratarse de solo equipo, el aumentar el número de procesos va en perjuicio del tiempo total de la simulación, debido a la sobrecarga que supone la comunicación entre los mismos.

4.3.3 Comparativa de tiempos en cluster Kasukabe

En una segunda fase de experimentación, decidimos ejecutar la simulación en el cluster Kasukabe.

En primer lugar analizamos la ejecución con un solo nodo:

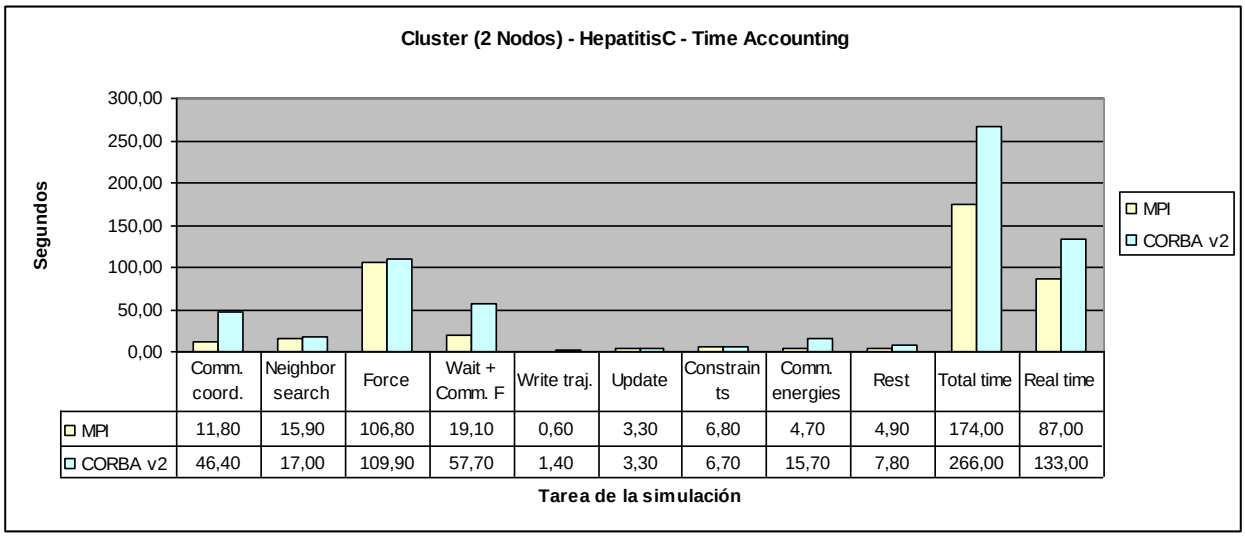


	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
MPI	0,00	15,40	103,80	0,00	0,30	3,50	6,50	0,00	3,50	133,00	133,00
CORBA v2	0,00	15,20	103,90	0,00	0,40	3,50	6,60	0,00	3,40	133,00	133,00

Figura 22: Hepatitis C. Tiempos en cluster con 1 nodo

Como se puede observar, el tiempo de ejecución es prácticamente el mismo en las dos implementaciones, porque realmente no se realiza ninguna comunicación, pero se observa que los tiempos han aumentado frente a la simulación en el equipo local (133 segundos, frente a 59 en local). Esto es debido a que las máquinas del cluster son menos potentes.

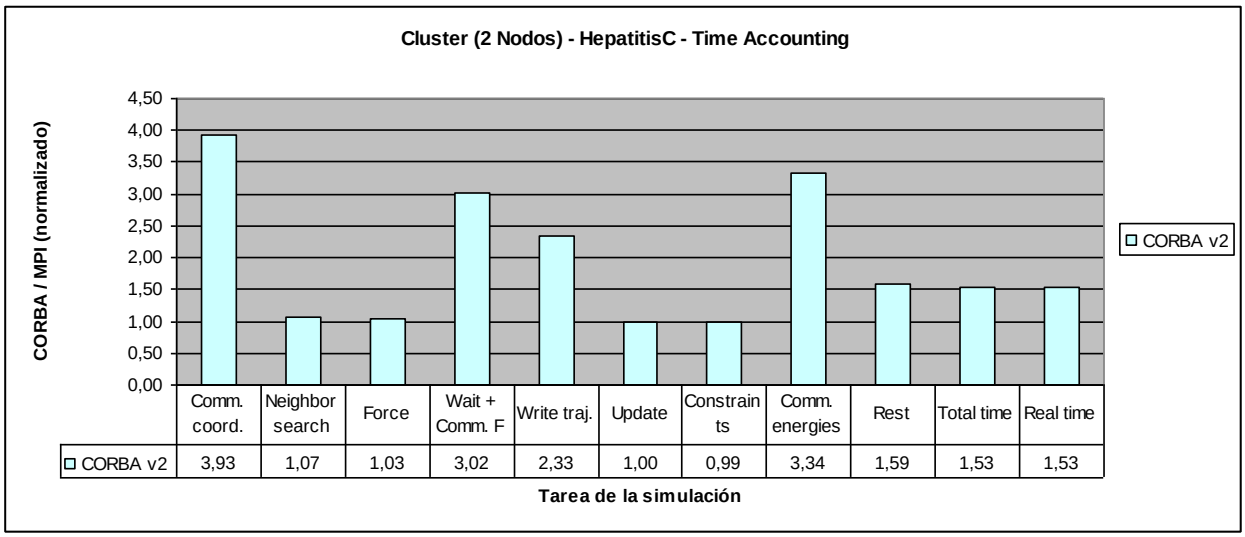
En la simulación con dos nodos obtuvimos:



	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
MPI	11,80	15,90	106,80	19,10	0,60	3,30	6,80	4,70	4,90	174,00	87,00
CORBA v2	46,40	17,00	109,90	57,70	1,40	3,30	6,70	15,70	7,80	266,00	133,00

Figura 23: Hepatitis C. Tiempos en cluster con 2 nodos

Si normalizamos los valores anteriores con respecto a MPI obtenemos la siguiente gráfica:

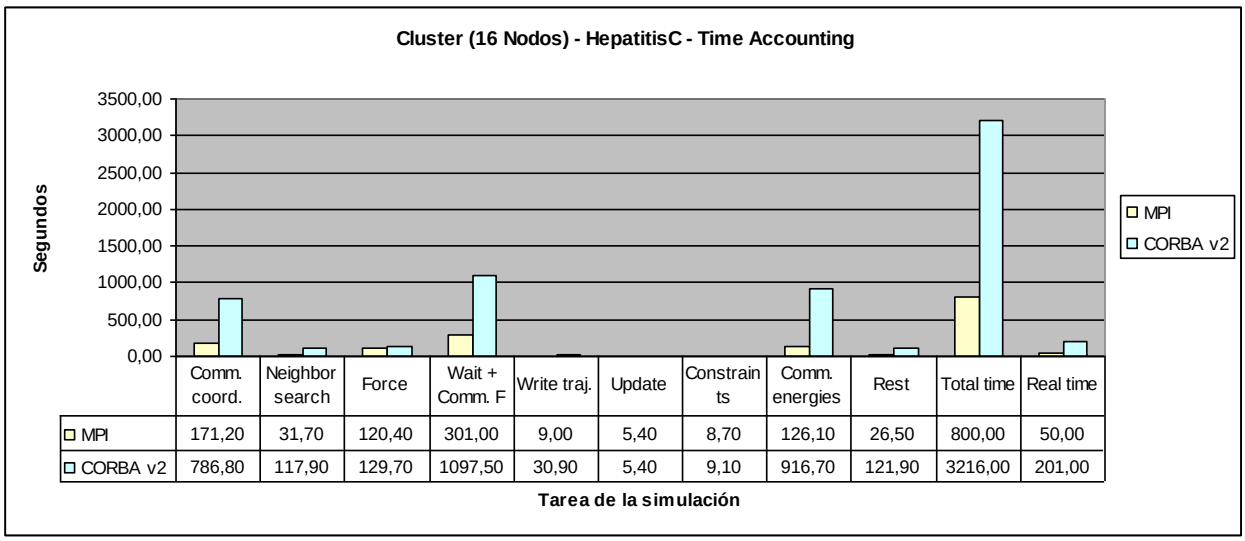


	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
CORBA v2	3,93	1,07	1,03	3,02	2,33	1,00	0,99	3,34	1,59	1,53	1,53

Figura 24: Hepatitis C. Tiempos en cluster con 2 nodos. Normalizado

En este caso se observa que el tiempo destinado a las comunicaciones es importante, pero a pesar de ello, en el caso de MPI el tiempo real de la simulación disminuye a 87, y en el caso de CORBA se mantiene en 133 segundos.
Proporcionalmente el tiempo real en la versión CORBA v2 es 1,53 veces superior al de MPI.

Si seguimos ampliando nodos, en el caso de 16 obtenemos los siguientes datos:



	Comm. coord.	Neighbor search	Force	Wait + Comm. F	Write traj.	Update	Constraints	Comm. energies	Rest	Total time	Real time
MPI	171,20	31,70	120,40	301,00	9,00	5,40	8,70	126,10	26,50	800,00	50,00
CORBA v2	786,80	117,90	129,70	1097,50	30,90	5,40	9,10	916,70	121,90	3216,00	201,00

Figura 25: Hepatitis C. Tiempos en cluster con 16 nodos

Si normalizamos los valores anteriores con respecto a MPI obtenemos la siguiente gráfica:

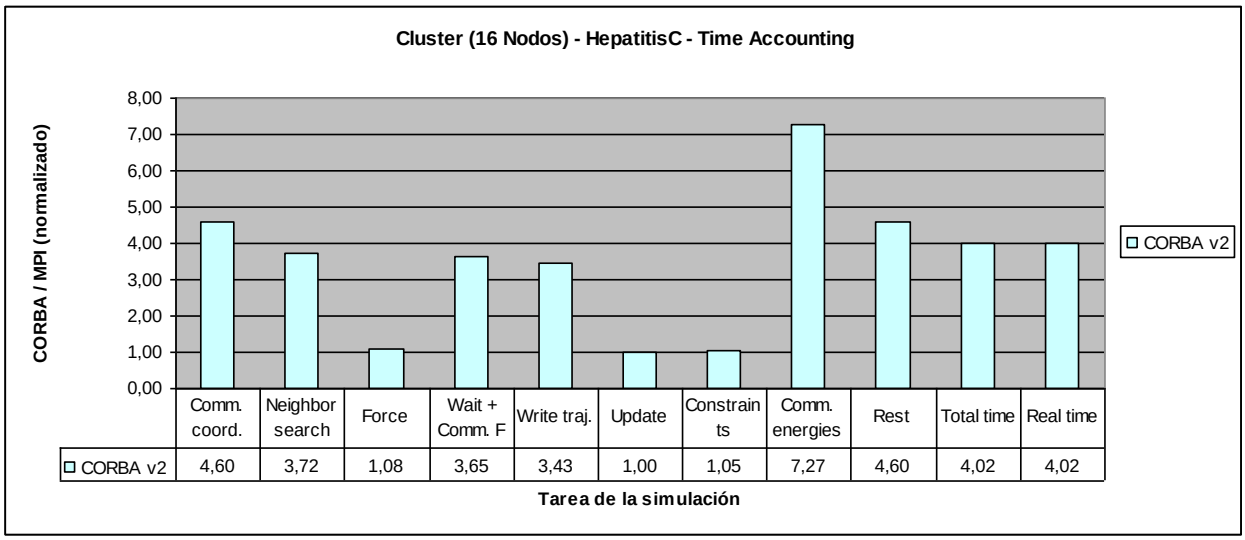
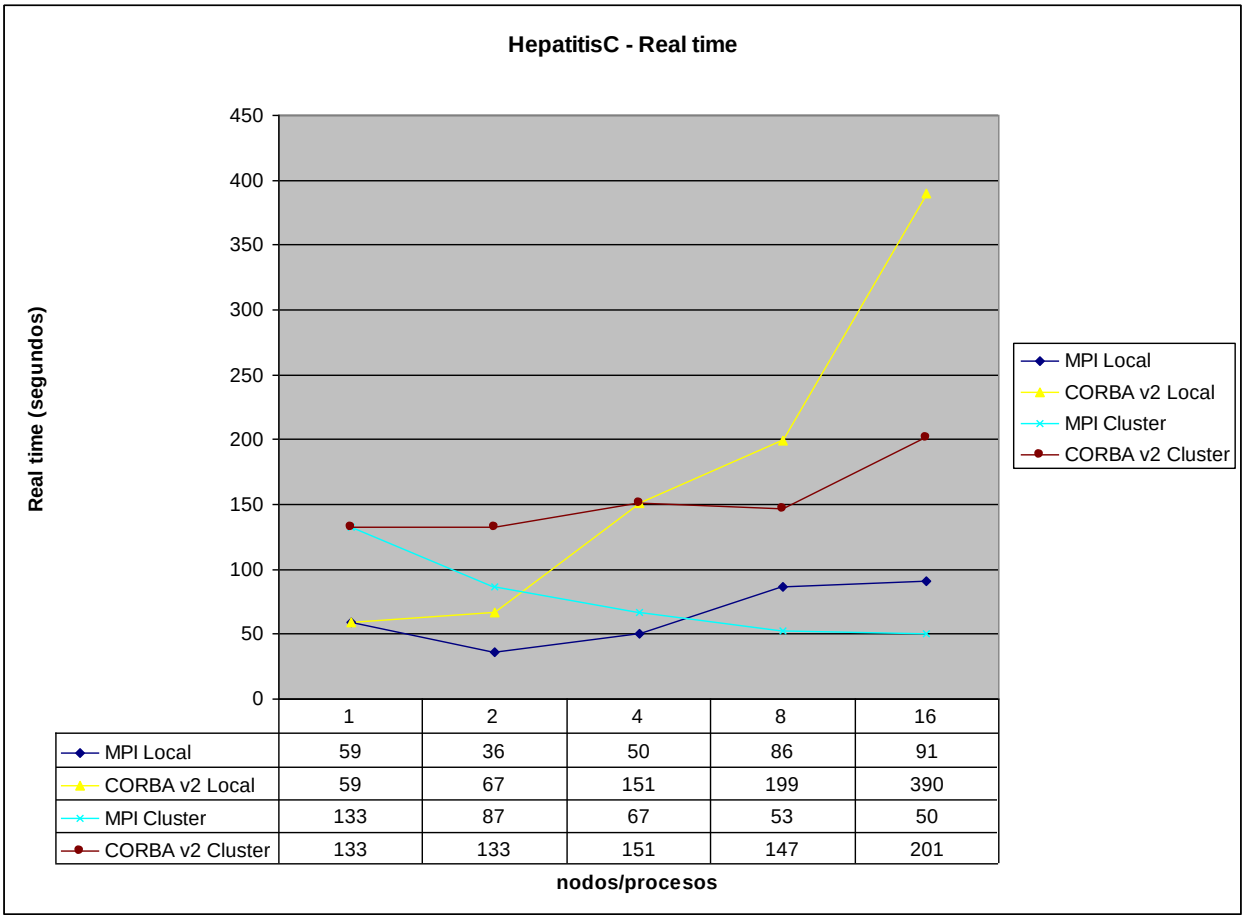


Figura 26: Hepatitis C. Tiempos en cluster con 16 nodos. Normalizado

En ese caso observamos, que usando muchos nodos, al tratase de un cluster el tiempo real disminuye para la versión MPI a 50. Los tiempos de la versión CORBA aumentan, pero en menor medida que en la molécula Speptide, debido al mayor tamaño de la molécula Hepatitis C. Con la molécula Speptide la versión CORBA v2 aumentaba el tiempo real en 10,77, mientras que con la molécula Hepatitis C aumenta en 4,02.

4.3.4 Resumen comparativa de tiempos

Como resumen, se analiza con la siguiente gráfica el tiempo real de ejecución de la simulación en cada uno de los equipos (local y cluster), para cada una de las implementaciones (MPI y CORBA v2) en función del número de nodos:



	1	2	4	8	16
MPI Local	59	36	50	86	91
CORBA v2 Local	59	67	151	199	390
MPI Cluster	133	87	67	53	50
CORBA v2 Cluster	133	133	151	147	201

Figura 27: Hepatitis C. Comparativa de tiempos

Se puede concluir, que la ejecución de MPI en local disminuye el tiempo para 2 y 4 nodos, pero aumenta a partir de 8.
La ejecución MPI en cluster supone una disminución del tiempo, incluso con 16 nodos, pero el porcentaje de reducción disminuye conforma aumenta el número de nodos.
Por último, la implementación CORBA no consigue disminuir los tiempos en ningún caso, pero, mientras que con la molécula Speptide se llegaban a multiplicar por 10 los tiempos frente a MPI, en la molécula Hepatitis C (de mayor tamaño) los tiempos se multiplican como mucho por 4 frente a la versión MPI.

4.4. Comparativa MPICH vs ORBit 2

En los anteriores estudios comparativos de rendimiento, se comprueba que hay un incremento generalizado de los tiempos de ejecución en los experimentos realizados con la nueva versión desarrollada con CORBA frente a la actual implementación con MPI.
Es por ello, que hemos querido hacer una comparativa de rendimientos entre ambas tecnologías, mediante un simple programa desarrollado en MPI, frente a otro similar pero desarrollado en CORBA.

Este programa de test, se limita ha realizar 100 envíos de 1.000.000 de bytes entre dos nodos.
En MPI esto se reduce a repetir 100 veces un *MPI_Send* en el nodo origen y un *MPI_Recv* en el destino con un buffer de 1.000.000 de bytes.
Para CORBA se ha definido el siguiente interfase:

```
module Examples {

	module ByteSeq {
		typedef sequence<octet> Chunk;

		interface Storage {
			void  set (in Chunk chunk);
			Chunk get ();
			void  exchange (inout Chunk chunk);
		};
	};
};
```

El tipo *sequence* representa el buffer y los métodos *set* y *get* serían equivalentes al *MPI_Send* y *MPI_Recv*.
El objeto servidor arranca y guarda su IIOR en disco, y posteriormente se inicia un cliente que realiza 100 veces la secuencia *set* + *get* sobre el buffer del servidor.

Este simple ejemplo demuestra que la versión CORBA conlleva una sobrecarga en las comunicaciones que podrían ser la causa del aumento de tiempos de la versión CORBA sobre la MPI.

El siguiente gráfico, refleja los tiempos de la prueba indicada:

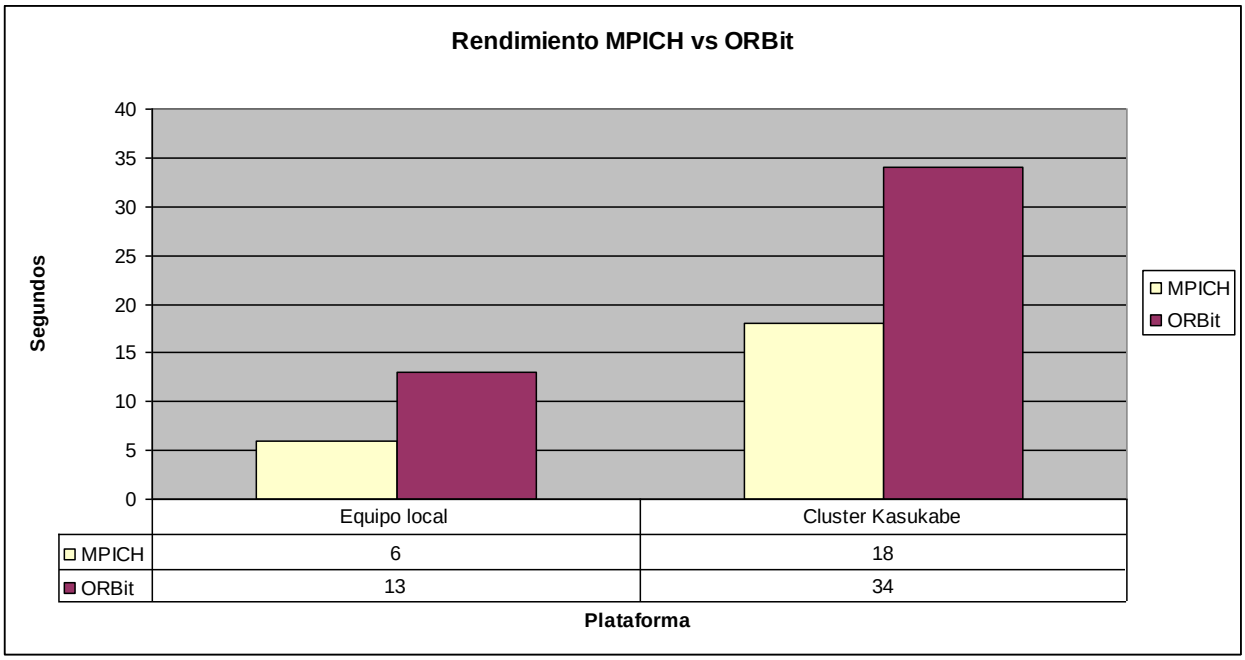


Figura 28: Rendimiento MPICH vs ORBit

Se evidencia por tanto una sobrecarga considerable en la implementación ORBit de CORBA frente a MPICH, probablemente debida al middleware (ORB) necesario para el reconocimiento de objetos remotos y la ejecución de métodos con grandes cargas de comunicaciones entre objetos.

Capítulo 5

Conclusiones y trabajos futuros

5.1. Conclusiones

Una vez concluido el proyecto, si analizamos los resultados con respecto a los objetivos marcados al comienzo, podemos sacar las siguientes conclusiones:

1. Se ha cumplido el primer objetivo del proyecto que consistía en reemplazar las llamadas a la librería de MPI por invocaciones CORBA. El proceso ha consistido en utilizar el middleware ORBit 2, una implementación de la especificación CORBA 2.4 en lenguaje C, para sustituir las llamadas a funciones MPI de GROMACS por funciones implementadas en una nueva librería desarrollada para este proyecto y que se apoya en este middleware.

2. Se ha cumplido el segundo objetivo marcado: la evaluación del rendimiento de GROMACS bajo CORBA. Se han empleado dos plataformas, un equipo local con procesador de doble núcleo y un cluster (Kasukabe) del laboratorio de informática de esta universidad, y se han considerado diferentes datos de entrada (simulaciones de la molécula Speptide y de la Hepatitis C), número de procesadores (entre 1 y 16), y dos implementaciones de la librería CORBANetwork (versión inicial v1, y versión optimizada v2).

3. Se ha cumplido el tercer objetivo marcado: el estudio de la adecuación de la implementación de GROMACS en CORBA para entornos de HPC (High-performance computing). La principal conclusión de este proyecto es que, en el marco de las arquitecturas y conjuntos de datos considerados, la implementación basada en CORBA no resulta viable dado que introduce una importante sobrecarga en las comunicaciones, la cual limita el rendimiento y escalabilidad de la implementación paralela del GROMACS.

5.2. Trabajos futuros

De cara al futuro, este proyecto podría completarse con otros trabajos. En concreto se podría realizar la implementación de una versión multiplataforma del kernel de GROMACS (nb_kernel112). Debido a que empleamos CORBA, esta implementación se podría desarrollar en Java/C++ u otro lenguaje de programación.

Resulta también interesante, evaluar el rendimiento de la plataforma en entornos de computación más fuertemente acoplados, como son las arquitecturas con múltiples núcleos. En estas plataformas el coste de las comunicaciones es mucho menor por lo que podría obtenerse un rendimiento positivo.

Los resultados obtenidos pueden ser extrapolados también a entornos de Cloud Computing, para los que el coste de las comunicaciones (ya sea por la propia infraestructura o el middleware de comunicación empleado) puede ser comparable a los considerados en este proyecto fin de carrera. Sería interesante realizar un estudio análogo al desarrollado en este trabajo con el fin de evaluar el rendimiento de la aplicación.

Capítulo 6

Presupuesto

6.1. Introducción

En este apartado, realizamos una estimación de tiempos y una valoración del coste económico del proyecto, simulando que se tratase de una iniciativa empresarial.

En función de estos costes se realiza un presupuesto en consecuencia con los mismos que es incrementado en un 30% con el fin de obtener beneficios.

Este porcentaje de beneficio sería revisable en función de las condiciones del mercado en el momento de presentar la oferta, y de la conveniencia o no de rebajar la oferta para fomentar posibles contrataciones futuras.

6.2. Estimación de tiempos

Una vez analizadas las tareas del proyecto, se ha estimado una duración total de 6 meses, en el caso de disponibilidad complementa de todos los recursos y una dedicación exclusiva a este proyecto, según el siguiente calendario:

ID	Task Name	Start	Finish	Duration
1	Estudio GROMACS	03/05/2010	20/05/2010	14d
2	Análisis implementaciones CORBA	21/05/2010	09/06/2010	14d
3	Desarrollo librería CORBANetwork	10/06/2010	10/08/2010	44d
4	Modificación GROMACS	26/07/2010	03/09/2010	30d
5	Primera fase experimentos	06/09/2010	14/09/2010	7d
6	Optimización librería CORBANetwork	15/09/2010	18/10/2010	24d
7	Segunda fase experimentos	19/10/2010	01/11/2010	10d
8	Estudio comparativo MPI vs CORBA	02/11/2010	15/11/2010	10d
9	Análisis y conclusiones	16/11/2010	17/11/2010	2d
10	Memoria y documentación	03/05/2010	30/11/2010	152d

Tabla 7: Proyecto. Estimación de tiempos

A continuación se muestra el diagrama de Gantt correspondiente:

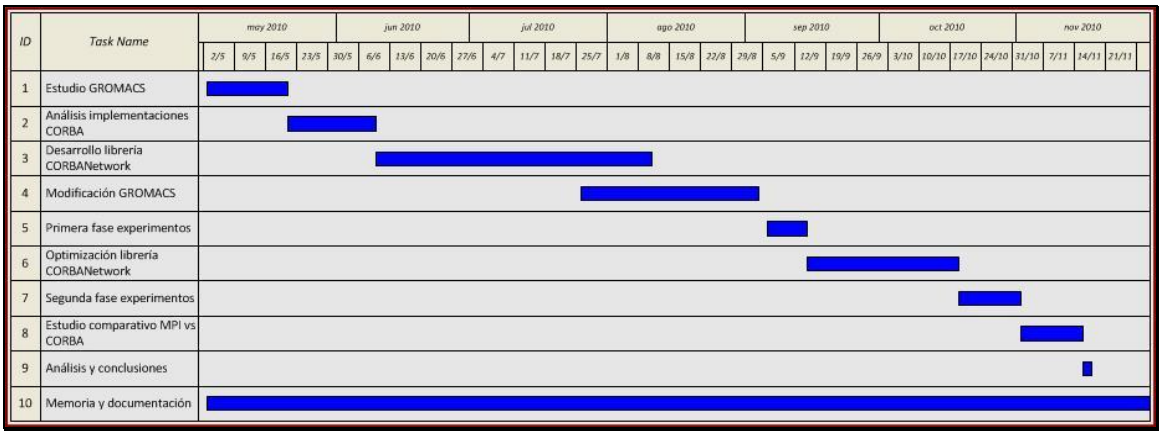


Diagrama 16: Proyecto. Diagrama de Gantt

6.3. Presupuesto

UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior

PRESUPUESTO DEL PROYECTO

1.- Autor: Fco. Javier Aliseda Casado

2.- Departamento: Informática. Área de Arquitectura y Tecnología de Computadores.

3.- Descripción del Proyecto: Diseño de un Simulador de Dinámica Molecular basado en CORBA
- Duración (meses) **6**
- Tasa de costes Indirectos: **IVA (20%)**

4.- Presupuesto total: **41.536 €**

5.- Desglose presupuestario (costes directos):

5.1. PERSONAL

Apellidos y nombre	N.I.F.	Categoría	Dedicación (hombres/mes) [a]	Coste hombre mes	Coste (Euro)
Fco. Javier Aliseda Casado		Ingeniero Técnico	6	2.694,39	16.166,34
David Expósito Singh		Ingeniero Senior	2	4.289,54	8.579,08
		Hombres/mes	**8**	**Total**	24.745,42

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

5.2 EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
PC desarrollo	1.600,00	100	6	60	160,00
PC pruebas	1.000,00	100	6	60	100,00
Cluster pruebas (16 nodos)	18.000,00	50	2	60	300,00
				Total	560,00

[d] Fórmula de cálculo de la Amortización: A/B * C * D

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

5.3 SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
Mantenimiento instalaciones	UC3M	600,00
Mantenimiento informático	UC3M	300,00
	Total	900,00

5.4 OTROS COSTES DIRECTOS DEL PROYECTO[e)]

Descripción	Empresa	Costes imputable
Material de oficina	FJAC	120,00
Dietas y desplazamientos	FJAC	240,00
Gastos varios	FJAC	60,00
	Total	420,00

[e)] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungibles, viajes y dietas, otros,…

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	24.745
Amortización	560
Subcontratación de tareas	900
Costes de funcionamiento	420
Costes Indirectos	5.325
Total	**31.951**

6.4. Oferta

Proyecto: Diseño de un Simulador de Dinámica Molecular basado en CORBA

El presupuesto total de este proyecto asciende a la cantidad de CUARENTA Y UN MIL, QUINIENTOS TREINTA Y SEIS euros.

Importe total del presupuesto: 41.536 € (IVA incluido)

Condiciones de la oferta:

A) El periodo de validez del presupuesto será de 30 días naturales desde la entrega del mismo.
B) La duración estimada será de 6 meses, suponiendo una disponibilidad completa del personal y disponibilidad total del equipamiento necesario.
C) Los pagos se fraccionará en los siguientes hitos:
   - 10% en el momento de la aceptación del presupuesto.
   - 35% a la entrega del análisis del sistema.
   - 35% a la entrega del código fuente.
   - 20% al cierre el proyecto.

   Los pagos se realizarán preferiblemente por transferencia bancaria.
D) Una vez realizadas las pruebas de aceptación y cerrada la entrega el proyecto, el sistema tendrá una garantía de 90 días naturales.
   Esta garantía cubrirá exclusivamente el mantenimiento correctivo del sistema.
   En caso de necesidad de mejoras o ampliaciones, se deberán tramitar con una nueva solicitud de servicio.

Leganés a 31 de Diciembre de 2010

El ingeniero proyectista

Fdo. Fco. Javier Aliseda Casado

Glosario

ASCII	*American Standard Code for Information Interchange*
CORBA	*Common Object Request Broker Adapter*
HPC	*High-performance computing*
MD	*Molecular Dynamics*
MPI	*Message Passing Interface*
OMG	*Object Management Group*
QSAR	*Quantitative Structure-Activity Relations*

Referencias

Internet:

- Sitio del proyecto GROMACS:
  http://www.gromacs.org/

- Desarrollo de aplicaciones distribuidas. Asignatura de la UC3M coordinada por David Expósito Singh (tutor de este proyecto):
  http://www.arcos.inf.uc3m.es/~dad/dokuwiki/doku.php

- Sitio sobre CORBA del OMG:
  http://www.corba.org

- Sitio del proyecto ORBit2 (implementación de CORBA usada en este proyecto):
  http://projects.gnome.org/ORBit2/

- Sitio del proyecto MPI (usado actualmente por GROMACS):
  http://www.mcs.anl.gov/research/projects/mpi/

- Páginas sobre GROMACS en el Pittsburgh Supercomputing Center:
  http://www.psc.edu/general/software/packages/gromacs/online/getting_started.html

- Páginas sobre Nanotecnología creada por el NSF (US National Science Foundation):
  https://nanohub.org/tools/biomolelab/wiki

- Portal de recursos para el estudio de macromoléculas biológicas:
  http://www.rcsb.org/pdb/home/home.do

Documentos:

- Benchmarks!
  David Expósito Singh (tutor de este proyecto)
  Presentacion_benchmarks.ppt

- Manual de GROMACS:
  gromacs4_manual.pdf

- Diagrama de flujo para realizar simulaciones de dinámica molecular con GROMACS:
  GROMACS_MD_Flowchart.pdf

- Documentación sobre ORBit2:
  orbit.pdf

Libros:

- M.L. Liu
  Computación Distribuida. Fundamentos y Aplicaciones.
  Pearson Educación, S.A. Madrid 2004

- Jorge E. Pérez Martínez
  Programación Concurrente
  Editorial Rueda, S.L. Madrid 1990

- M. Waite, S. Prata y D. Martin
  Programación en C. Introducción y conceptos avanzados.
  Ediciones Anaya Multimedia, S.A. Madrid 1991

ANEXOS

Anexo I. Manual de Usuario

Instalación

En el CD incluido con el proyecto se incluye el software requerido para su instalación en equipos compatibles i386 y sistema operativo Linux.
Los pasos a realizar serían:

1) Instalación de la librería FFTW, requerida por GROMACS para cálculos de la transformada de Fourier:
   - Crear la carpeta: /home/<username>/fftw
   - Descargar y descomprimir el paquete con los fuentes de FFTW-3.1.2 (incluido en la carpeta "*software*" del CD)
   - Ejecutar:

```
./configure --prefix=/home/<username>/fftw --enable-float --enable-threads
make
make install
export CPPFLAGS=-I/home/<username>/fftw/include
export LDFLAGS=-L/home/<username>/fftw/lib
```

2) Instalación de la implementación MPICH de MPI y servidor ssh desde el paquete binario:

```
sudo apt-get install mpich-bin
sudo apt-get install libmpich1.0-dev
sudo apt-get install openssh-server
```

3) Instalación de la implementación ORBit2 de CORBA desde el paquete binario:

```
sudo apt-get intall orbit2
sudo apt-get intall liborbit2-dev
```

4) Instalación de GROMACS con soporte de interoperabilidad CORBA:
   - Crear la carpeta: /home/<username>/gromacs
   - Descargar y descomprimir el paquete con los fuentes de GROMACS (incluido en la carpeta "*desarrollo*" del CD)
   - Adaptar el script "*compila-gromacs*" de la carpeta "*desarrollo*" del CD a la estructura de directorios del equipo de instalación y proceder a su ejecución:

```
#!/bin/sh

cd /home/javier/pfc/gromacs

export CPPFLAGS="-I/usr/local/include -I/home/javier/pfc/fftw/include -DORBIT2=1
       -pthread -I/usr/include/orbit-2.0 -I/usr/include/glib-2.0
       -I/usr/lib/glib-2.0/include -I/home/javier/pfc/corbanetwork"

export LDFLAGS="-L/home/javier/pfc/corbanetwork -L/home/fortran/localinstall/lib
       -L/home/javier/pfc/fftw/lib -Bdynamic -lcorbanetworkpfc -Bstatic -pthread
       -lORBitCosNaming-2 -lORBit-2 -lgthread-2.0 -lrt -lgobject-2.0 -lglib-2.0"

./configure --enable-mpi --prefix=/home/javier/pfc/gromacs --enable-float
       --enable-threads --enable-sse

make
make install

export PATH="/home/javier/pfc/gromacs/bin:$PATH"
export LD_LIBRARY_PATH="/home/javier/pfc/corbanetwork"
```

Generación ficheros simulación

Para la ejecución de una simulación hay que comenzar por la generación de los ficheros de entrada para la simulación.

Los pasos para la generación de estos ficheros están explicados en el apartado *"Generación de ficheros de entrada para la simulación"*.
Para las pruebas realizadas en este proyecto se ha utilizado el script "*genSpeptide*" incluido dentro de la carpeta "*moleculas*" del CD, cuyo contenido es el siguiente:

```
# Genera los ficheros necesarios para experimentar con gromacs
read -p "Generate a GROMACS topology. Use option 0. Press any key to start"
pdb2gmx -f speptide.pdb -p speptide.top -o speptide.gro

read -p "Enlarge the box. Press any key to start"
editconf -f speptide -o -d 0.5 -c

read -p "Solvate. Press any key to start"
genbox -cp out -cs -p speptide -o b4em

read -p "Generate index file. Use option q. Press any key to start"
make_ndx -f b4em

read -p "Generate mdrun imput file for EM. Press any key to start"
grompp -v -f em -c b4em -o em -p speptide

read -p "Run simulation for EM. Press any key to start"
mdrun -v -s em -o em -c after_em -g emlog

read -p "Perform a short MD run with position restraints. Press any key to start"
grompp -f pr -o pr -c after_em -r after_em -p speptide
```

Ejecución en equipo local

Posteriormente, para la ejecución se utiliza el script "*lanzador*" situado en la misma carpeta y que a su vez llama a varios scripts para simular la ejecución con distinto número de procesos y almacenar los resultados.

Para simular 16 procesos y obtener los resultados en el fichero "*salida16*" el script "*lanzador16*" realiza lo siguiente:

```
#!/bin/sh

rm *.ref
rm *.log
rm *#

mpirun -np 16 /home/javier/pfc/gromacs/bin/mdrun -v -s pr -e pr -o pr
       -c after_pr -g prlog 2> salida16
```

Ejecución en cluster Kasukabe

El cluster Kasukabe del laboratorio de informática de universidad, necesita que se le indiquen los parámetros de los trabajos a ejecutar antes de lanzamiento.
Para automatizar las pruebas, se han definido estos parámetros en diferentes scripts en función del número de nodos a utilizar.

Como ejemplo, para realizar una simulación con 16 nodos y obtener los resultados en el fichero "*salida16*" el script "*lanzador16.qsub*" realiza lo siguiente:

```
#PBS -N gromacs16
#PBS -q long
#PBS -o salida16
#PBS -e error16
#PBS -l nodes=16
#PBS -l walltime=01:00:00

cd $PBS_O_WORKDIR

echo "Contenido de $PBS_NODEFILE: "
cat $PBS_NODEFILE

rm *.ref
rm *.log
rm *#

mpirun.mpich -np 16 -machinefile $PBS_NODEFILE
/datos_nfs/pfc_aliseda/gromacs/bin/mdrun -v -s pr -e pr -o pr -c
/datos_nfs/pfc_aliseda/speptide/after_pr -g
/datos_nfs/pfc_aliseda/speptide/prlog >
/datos_nfs/pfc_aliseda/speptide/misalida16

exit
```

Para encolar el trabajo en el gestor de colas ejecutaríamos el comando:

qsub lanzador16.qsub

Para comprobar el estado de los trabajos utilizaremos el comando:

qstat -a